1965

Hybrid control system simulation.

Holloway, Lowell J.

Monterey, California: U.S. Naval Postgraduate School

http://hdl.handle.net/10945/12303

HYBRID CONTROL SYSTEM SIMULATION

LOWELL J. HOLLOWAY

93

HYBRID CONTROL SYSTEM SIMULATION

* * * * *

Lowell J. Holloway

HYBRID CONTROL SYSTEM SIMULATION

by

Lowell J. Holloway

Lieutenant Commander, United States Navy

Submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING

United States Naval Postgraduate School
Monterey, California

1 9 6 5

HYBRID CONTROL SYSTEM SIMULATION

by

Lowell J. Holloway

This work is accepted as fulfilling
the thesis requirements for the degree of
MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING
from the
United States Naval Postgraduate School

ABSTRACT

Hybrid control systems are becoming increasingly important today due primarily to the development of the high speed digital computer. This work involves the hybrid simulation of a relatively simple second order analog system using a short word length digital computer for the computation of the control law for the analog system. Particular attention is paid to the effects of the analog-digital interface on the analog system response. A method is developed whereby the digital computer can be programmed to minimize these interface effects.

The simulation was carried out in the Digital Control Laboratory at the U. S. Naval Postgraduate School using the CDC 160 digital computer and the Pace TR-20 analog computer with associated equipment. Although the specific solution of the interface problem is directed towards the above equipment, the general method of attack on the interface problem is applicable to many similar type hybrid systems.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF SYMBOLS

A/D	Analog-to-digital conversion process.
D/A	Digital-to-analog conversion process.
ADC	Analog-to-digital converter.
DAC	Digital-to-analog converter.
$(xxx)_{10}$	xxx is a number expressed in the decimal number system.
$(xxxx)_8$	xxxx is a number expressed in the octal number system.
$(xxxx)_2$	xxxx is a number expressed in the binary number system.
$(\overline{xxxx})_2$	The one's complement of the binary number xxxx.
X_i	The ith state variable of an analog system.
a_i	The ith feedback coefficient.

CHAPTER I

INTRODUCTION

The combined use of analog and digital computers in the simulation of complex systems has increased manyfold in recent years, due primarily to the increase in speed of computations of the digital computers. These hybrid simulation schemes combine the parallel computation features of the analog computer with the accuracy and versatility of the digital computer.

This study was undertaken with the purpose of developing a series of programs and subroutines which could be used for the simulation of a large number of hybrid control systems using a digital computer with a short word length and limited arithmetic computational ability. As the study developed, it became apparent that one of the more important considerations was the analog-digital interface and its possible effect on system response. As a result, the major effort was directed towards the detailed study of a simple second order hybrid system simulation in order to investigate these effects in terms of a specific system.

Chapter two gives a brief description of the equipment used in the simulation study; Chapter three describes the general considerations which must be considered in any hybrid control scheme, and Chapter four deals with the actual simulation study itself.

Towards the end of developing general programs, a subroutine (Appendix II) was developed for general use which provides for the introduction of computational constants into the digital computer in a format compatible with the analog-digital conversion equipment. Also, the digital computer program developed for the system simulation is applicable to any second-order analog system where the digital computer is used to compute a control value consisting of a linear combination of the states of the

analog system. Also, the general method of attack on the limitations imposed by the A/D interface is applicable to hybrid control systems in general although the specific procedures may vary with each specific system.

CHAPTER 2

DESCRIPTION OF EQUIPMENT

2.0 GENERAL.

The experimental phases of this thesis were carried out in the Digital Control Laboratory, U. S. Naval Postgraduate School. System simulation was accomplished using the following equipment:

A. The Control Data Corporation 160 digital computer.

B. The Control Data Corporation 168 arithmetic unit.

C. The Pace TR-20 analog computer.

D. A 12-bit, multi-channel analog-to-digital and digital-to-analog conversion unit (ADC-DAC).

Brief descriptions of the above equipment are given in the following paragraphs.

2.1 CONTROL DATA CORPORATION 160 DIGITAL COMPUTER.

The CDC 160 digital computer is a electronic computer controlled by an internally stored program in sequential locations. Memory capacity is 4096 12-bit binary words of magnetic core storage, with a storage cycle time of 6.4 microseconds. Instructions are executed in one to four storage or memory cycles with the time required for execution varying from 6.4 to 25.6 microseconds. The average instruction execution time is 15 microseconds.

A CDC 160 computer word is made up of 12 binary digits. The bits within a computer word are numbered from 0 to 11, right to left, i.e.,

11	10	09	08	07	06	05	04	03	02	01	00

All arithmetic is binary, one's complement notation. Although the computer

operates in the binary system, the octal representation of a binary number is more convenient. The CDC 160 word can then be expressed as four octal digits, i.e.,

011	110	001	011	binary
3	6	1	3	octal

All positive numbers must have a "0" in bit 11; all negative numbers must have a "1" in bit 11. In octal notation, numbers from $(0000)_8$ thru $(3777)_8$ are positive; numbers from $(4000)_8$ thru $(7777)_8$ are negative.

All programs were compiled using the One Sixty Assembly Program (OSAP). The OSAP output consists of a side-by-side listing of the machine language instruction code and a alpha-numeric program listing. The CDC machine language instruction word is divided into a six bit function code (F code) followed by a six bit execution address (E field). Most instructions follow this 12-bit pattern and thus require only one word of storage; however, certain expanded instructions require 24-bits thus using two words of storage. The first word of such a two word instruction consists of the six-bit function code followed by a six-bit execution address which is always zero. The succeeding 12-bit word contains the address or operand (G code) depending on the instruction.

The CDC 160 computer communicates with external or peripheral equipment by means of external function codes. A detailed description of these external function codes is given in Appendix II.

The CDC 160 computer is very limited in its arithmetic computation capability. Twelve bit addition and subtraction is accomplished in two or three memory cycles, depending upon the address mode used. Multiplication and division can only be accomplished by successive 12-bit addition or

subtraction, respectively, resulting in excessive programming and execution time requirements.

2.2 CONTROL DATA CORPORATION 168 ARITHMETIC UNIT.

The peripheral CDC 168 arithmetic unit provides the CDC 160 computer with the capability of double precision addition and subtraction plus single or double precision multiplication and division of data. The CDC 168 is addressed by the CDC 160 with external function codes (See Appendix II) which select the unit and specify which operation is to be performed. The computer then transmits the proper number of words in the correct format to make up the arguments for the operation selected. The CDC 168 proceeds to compute the result and awaits the computer request for results.

In the programs which follow, the CDC 168 is used in the single precision multiply and divide modes. Single precision (short) multiplication uses an 11-bit multiplier and an 11-bit multiplicand, providing a 22-bit product. Single precision (short) division requires a 22-bit dividend and a 11-bit divisor yielding a 11-bit quotient. This operation requires from 145 to 180 microseconds while the short multiply operation requires 120 microseconds. The 22-bit arguments are formed from two 12-bit CDC 160 words using the least significant 11-bits of the two words while the most significant bit in each word is considered as a sign bit. Appendix I gives several examples of the short multiply and divide operations. Double precision multiplication (22 bits x 22 bits) and division (44 bits $\div$ 22 bits) are available at the expense of doubling the time per operation.

2.3 ANALOG-TO-DIGITAL AND DIGITAL-TO-ANALOG CONVERSION EQUIPMENT.

A 12-bit, multi-channel analog-to-digital and digital-to-analog conversion package was used to convert analog voltages to digital numbers and

to convert digital numbers into analog voltages. Analog voltages to be converted to digital numbers must lie in the zero to minus ten volt range. Similarly, digital numbers converted to analog voltages result in voltages between the same limits. This restriction of voltages to and from the D/A conversion equipment will usually require the application of bias voltages to the actual analog system quantities.

For analog computer simulation studies, the necessary biasing is accomplished by the application of a five volt bias to the actual system voltages in an operational amplifier used as a summer. The inherent sign inversion will result in the proper input to the ADC. For example, if X represents the analog quantity to be converted to a digital number, then the actual input to the ADC is -(X +5) which will lie in the conversion range if $|X| \leq 5$ volts. Table 2-1 is a analog voltage-digital number conversion table for the ADC-DAC package. It should be noted that positive, unbiased voltages result in positive digital numbers (bit 11 is "0"). The same sign correspondence is also applicable for negative values.

Unbiased Analog Voltage (volts)	Biased Analog Voltage (volts)	Digital Number (octal)
-5	0	4000
-4	-1	4632
-3	-2	5463
-2	-3	6315
-1	-4	7146
0	-5	0000 or 7777
1	-6	0631
2	-7	1462
3	-8	2314
4	-9	3144
5	-10	3777

Table 2-1. Analog Voltage-Digital Number Conversion Table.

Approximately 100 microseconds is required for each analog-to-digital number conversion and approximately 20 microseconds is required for digital-

to-analog conversion.

Accessory equipment included with the DAC-ADC package includes a common five-volt bias source, a "input disable" jack and a relay which is controllable by means of an external function code from the CDC 160 computer. The relay operation is explained in Appendix II. The common bias source provides a common bias to all ADC inputs and DAC outputs thus eliminating individual bias adjustment problems and errors.

The "input disable" jack is connected by means of an AND-gate to the "input ready" line of the CDC 160 computer input capable and provides a means of delaying the CDC 160 computer by an external timing device. To input a digital word from the ADC, the CDC 160 computer sends an "input request" to the ADC at which time the ADC converts the analog voltage to a digital number on the selected A/D channel. Upon completion of the conversion process, the ADC sends an "input ready" signal to the computer after which the computer will input the digital words. If, however, the "input ready" is held at ground level by means of an external device, the "input ready" signal to the computer is delayed until the external device drops the "input disable" to -13 volts. Thus, external sample timing control may be accomplished via the "input disable" jack. Note, however, that when the "input ready" line is released by the external clock, a stale analog sample value will be transmitted to the digital computer and should be discarded.

2.4 PACE TR-20 ANALOG COMPUTER.

The Pace TR-20 analog computer is a solid state analog computer employing twenty operational amplifiers. Saturation levels on these amplifiers are ($\pm$) ten volts. Standard analog computational techniques are applicable to this computer. External connection to the "operation-reset" relays is provided which permits remote control of the computer from the CDC 160 digital computer using the accessory relay of the ADC-DAC equipment (See Appendix II).

CHAPTER 3

GENERAL HYBRID CONTROL SYSTEM CONSIDERATIONS

3-0. GENERAL LIMITATIONS IMPOSED BY HYBRIDIZATION.

If a digital computer is to be used in the control of an analog system, then the following factors may, in general, affect the system response:

A. The inherent time delay required to convert a sequence of analog quantities to digital numbers.

B. The inherent time delay imposed by the serial nature of computation in a digital computer (as opposed to parallel computation in an analog sense).

C. The manner in which computational constants are entered into the computer.

D. The method used to obtain sample timing control if the analog inputs to the digital computer are to be sampled at regular intervals.

E. The inherent magnitude limitations imposed by the A/D conversion process.

Each of these factors must be considered in the implementation of hybridization, and are discussed more fully below.

3.1 A/D CONVERSION TIME DELAY.

The analog-to-digital conversion of an analog voltage cannot be done instantaneously but requires a finite time dependent upon the type of conversion process used and the specific equipment package. As explained in Chapter 2, the successive approximation type converter used for this project requires about 100 microseconds for the conversion process. In many applications, a simultaneous sampling of several analog quantities is desired; however, due to this finite conversion time, simultaneous sampling cannot be achieved by the A/D converter. The seriousness of this problem is determined by the number of "simultaneous" samples desired and the accuracy of

solution desired.

As will be explained in Chapter 4, four channels of A/D input were sampled in succession for a particular simulation problem with negligible effect noticeable due to non-simultaneous sampling. However, if a large number of inputs are desired, time delay effects may become significant. In this case, an external sample-and-hold device will undoubtedly be necessary to sample all inputs simultaneously and hold the individual samples until the ADC completes conversion on all channels.

3.2 COMPUTATION TIME DELAY.

A digital computer must necessarily perform arithmetic operations in a step-by-step or serial manner. When used for hybrid control purposes, the usual application involves a series of arithmetic operations on sampled input quantities and the "outputting" of the result. Due to the serial nature of computation, a finite time exists between input of samples and output of result. This time delay is a function of the cycle time of the specific computer and the complexity of computation.

Efficient computer programming is obviously required to minimize these effects and in some applications a prediction scheme may be called for if extreme accuracy is desired.

3.3 ENTRY OF CONSTANTS INTO THE COMPUTER.

Regardless of the type of computation required in a hybrid control computer, certain constants must be entered into the digital computer. If the computer has a floating point capability, constant entry does not pose a serious problem. However, for a fixed point machine (such as the CDC 160), the format in which these constants are entered into the computer may have a pronounced effect on the amount of programming required.

Appendix I explores this problem in detail for the CDC 160 computer which requires access to an external arithmetic unit for multiply and divide operations. To minimize the computation time delay (Paragraph 3.2) and to provide ease of programming, the method of constant entry used herein is to convert all constants to the number system of the analog-digital converter (Table 2-1). All arithmetic computations are then done in this "psuedo-number" system. See Example 4, Appendix I. Specifically, the method used herein is:

A. Enter constants into selected storage cells of the 160 computer using normal decimal to octal conversion procedures to obtain the octal representation of a decimal number. The format of entry is $(x.xxx)_8$, i.e., the radix point is implied after the most significant octal digit. The number $(2.5)_{10}$ would be entered as $(2400)_8$.

B. Conversion of the constants from normal octal notation to the "psuedo-number" system of the ADC (Table 2-1). This is done by means of a subroutine (See Appendix III, Subroutine KMOD), and is accomplished by multiplying the constant to be converted by $(0631)_8$ which corresponds to unity in the psuedo-number system. For example, the number $(2.5)_{10} = (2400)_8$ is converted to $(1776)_8$.

C. All further arithmetic operations are done in this new number system.

As shown in Example 4, Appendix I, every multiplication must be followed by division by unity. This seems somewhat cumbersome, but allows the use of a wide range of constants and a minimum of programming to obtain a single 12-bit number from the 168 arithmetic unit.

The above method of constant entry is tailored specifically for the CDC 160 computer. The method of attack on this problem may be quite different for a different digital computer. Figure 3-1 is a flow chart for a general

program which computes an output quantity which is a linear combination of two sample values. It should be noted that constant conversion is required only once and may be done outside of the sample loop.

3.4 SAMPLE TIMING CONTROL.

The digital computer will normally be required to sample the inputs at regular intervals. A method must be established to obtain this sample interval and can be done either internally or externally. Internal timing can be programmed into the computer using the known memory cycle time of the computer and having the computer perform some fixed time operation repeatedly until the desired time delay is accomplished. The "time delay loop" is entered after the desired computations are completed in each sample cycle. (See Figure 3-1). This method suffers from two disadvantages. First, it is a rather inefficient method of use of a digital computer and secondly, the actual time delay is dependent upon variations in computation time of arithmetic operations performed on the sample inputs, i.e., if the amount of computation varies from cycle to cycle, then the actual time interval between successive samples will also vary.

Alternatively, an external timing device can be used to either "hang up" the computer (such as the "input disable" explained in Chapter 2) or allow time-sharing of the computer with other tasks. External timing will provide the most accurate timing reference since the sample timing is then independent of the computation program execution time.

Internal timing control was used for the simulation scheme used in Chapter 4, however, provisions were included in the program for external timing control, if desired.

3.5 MAGNITUDE LIMITATIONS IMPOSED BY THE ADC.

Associated with each ADC is a magnitude limitation imposed on the analog

quantities which must undergo conversion. This is a physical equipment limitation and will depend upon the specific ADC-DAC conversion package. This limitation not only limits the magnitude of the analog quantities to be converted, but also limits the arithmetic computation range in the digital computer <u>if</u> all arithmetic operations are carried out in the analog-digital psuedo-number system. This problem is covered in detail in Chapter 4.

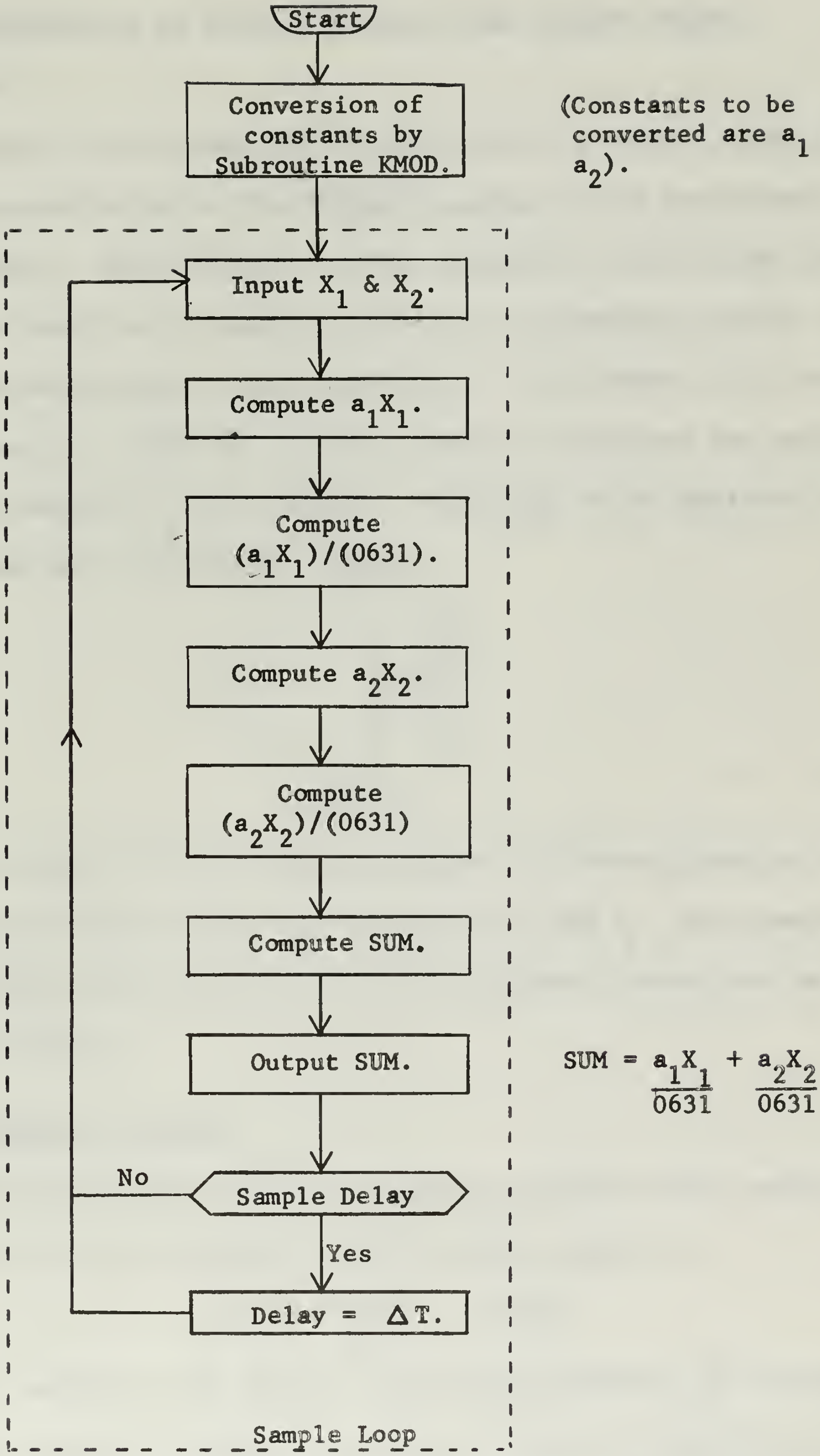


Figure 3-1. General Program Flow Chart For Two Sample Inputs.

CHAPTER IV

SIMULATION OF AN OPTIMAL DISCRETE-TIME CONTROL SYSTEM

4.0 GENERAL

The presence of an analog-digital interface in a control scheme may have a pronounced effect on the dynamic response of the system under certain conditions. These effects are due primarily to the A/D-D/A conversion process and must be taken into account if successful control is to be accomplished over widely varying conditions. This chapter will describe the simulation of a specific system in order to illustrate the possible limitations imposed by hybridization. The system to be simulated is governed by the ordinary differential equation

$$\dot{\underline{X}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \underline{X} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} U \qquad (4.1)$$

$$\underline{X}(0) = \underline{C}$$

The CDC 160 computer will be used to compute the forcing function, U, based on the sampled values of the state variables X_1 and X_2. The simulation will be accomplished in terms of both the regulator problem and the non-autonomous problem.

4.1 THE REGULATOR PROBLEM.

Bertram [1] has shown that for a sampling interval of one second, a time-optimal forcing function, U, for the above system is

$$U(k) = - X_1(k) - 1.5X_2(k) \qquad (4.2)$$

where $X_i(k)$ represents the value of the state variable X_i at the kth sampling instant and U(k) represents the control value to be applied to the system between the kth and k+1st sample. This control law will force all of

the state variables to zero in minimum time. The system is shown in block diagram form in Figure 4-1.

The continuous plant will be simulated on the PACE TR-20 analog computer. The CDC 160 digital computer will be used to sample the state variables X_1 and X_2 at one second intervals via the ADC and compute the forcing function U(k) which then will be transmitted to the continuous plant via the DAC. The DAC acts as a zero-order hold for the continuous plant. A block diagram of the hybrid system simulation is shown in Figure 4-2.

4.1.1 LIMITATIONS IMPOSED BY HYBRIDIZATION.

Simulation of the analog portion of the system is done in a conventional manner, the only restriction being that the maximum analog values of the state variables lie within ($\pm$) five volts (prior to biasing) so as to remain within the voltage limitations imposed by the ADC. This limitation, at most, will impose a conventional analog computer magnitude scaling requirement. The bias voltages must be applied to ensure that the analog state variable voltages lie within the zero to minus ten volt range for A/D-D/A conversion.

The CDC 160 computer program appears to be relatively simple and straightforward following the flow chart shown in Figure 3-1. However, if a wide range of initial conditions and state variable values is to be considered, then additional limitations impose themselves on the programming of the computer. Consider, for example, the formation of the product

$$(-1.5)_{10} \times (4)_{10} = (-6)_{10}$$

The 160 computer generates this product as

$$(6632)_8 \times (3144)_8 = (7153)_8 = (-1)_{10}$$

This inconsistency can be attributed to the fact that the magnitude of the

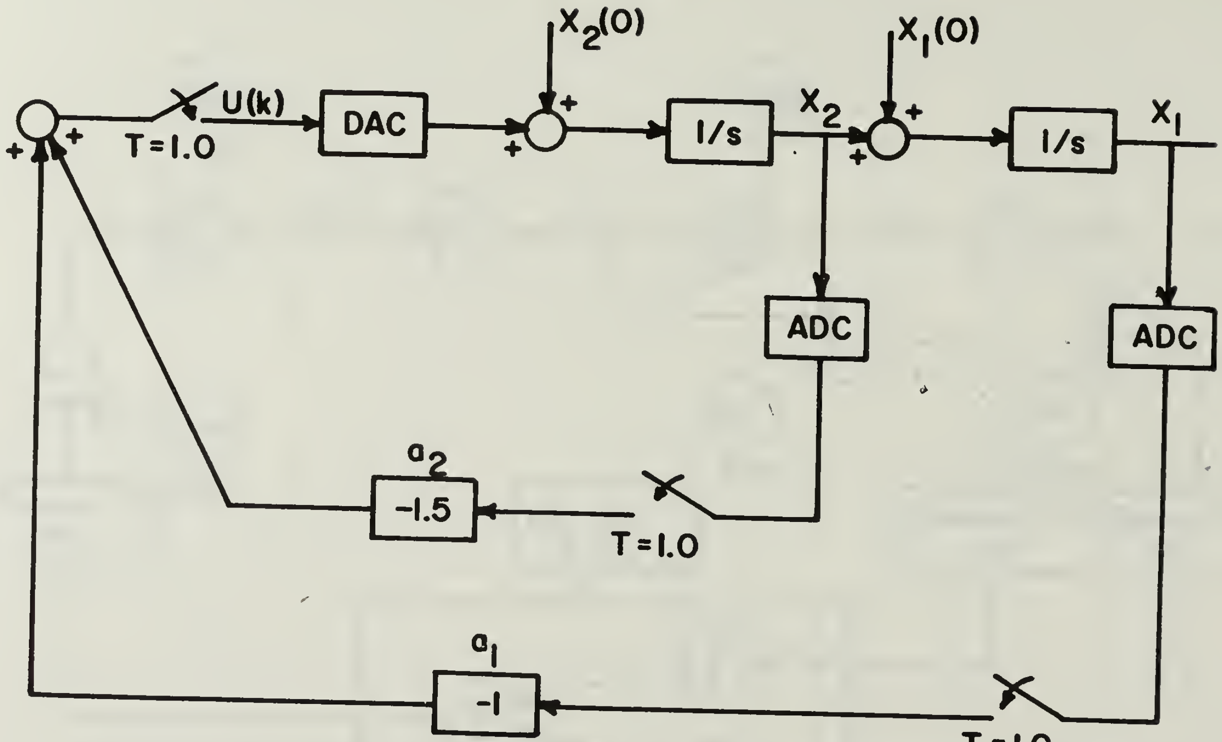


FIGURE 4-1 BLOCK DIAGRAM OF SYSTEM TO BE SIMULATED

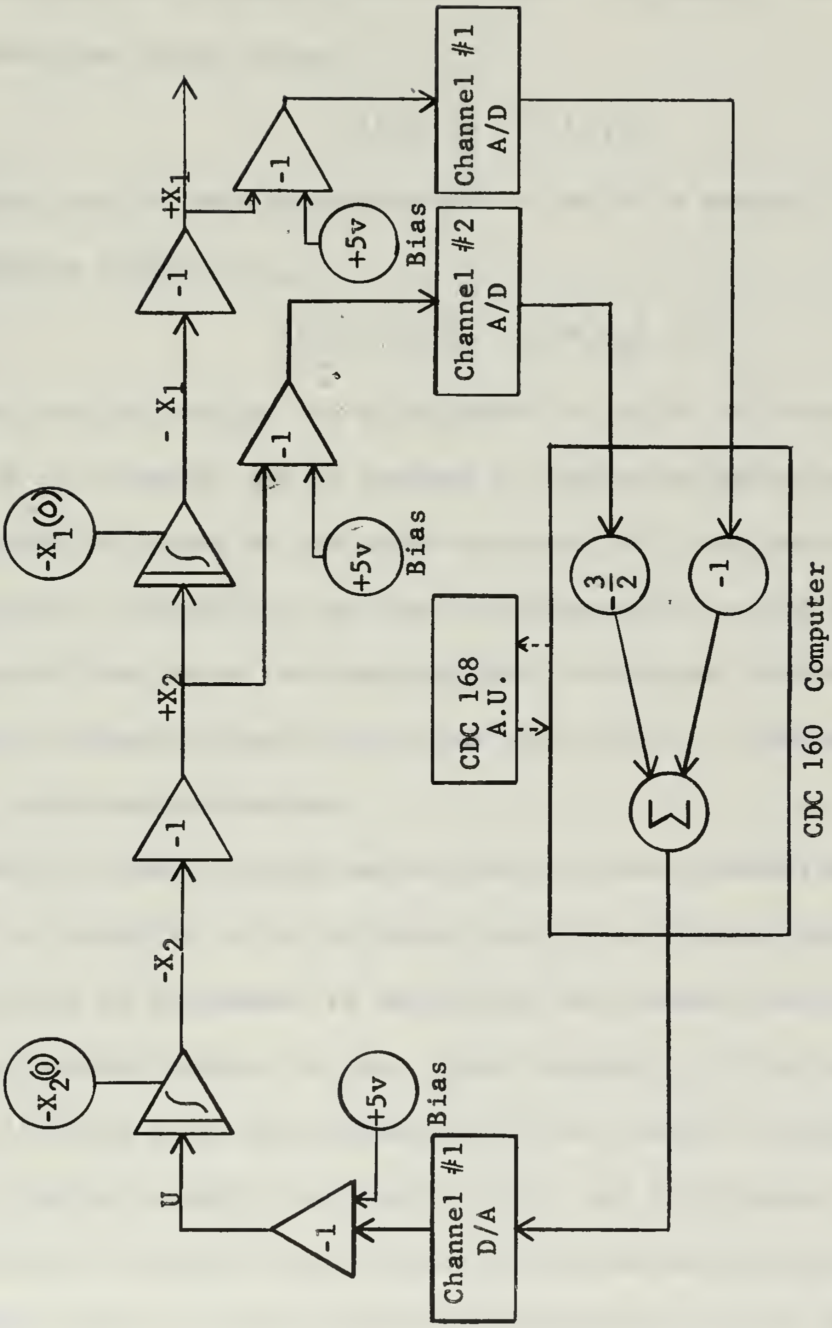


Figure 4-2 Block diagram of Simulation Scheme

desired product is greater than five while the "pseudo-number system" associated with the analog-digital interface is consistent only for numbers of magnitude less than five. The desired product (-6) has "overflowed" the number system and erroneous results occur. In general, for the product a_iX_i, overflow occurs unless

$$|X_i| \leq 5/|a_i| \tag{4.3}$$

A similar type of overflow occurs when a sum of a series of numbers exceeds the overflow limit, i.e.,

$$|a_1X_1 + a_2X_2 \quad \ldots + a_nX_n| \leq 5 \tag{4.4}$$

This overflow problem can be attacked in one of two ways. First, a suitable scale factor may be applied to the analog values such that the maximum scaled values of the state variables will not cause overflow. This method is simple but can lead to inaccuracies as the state variables tend toward zero where the analog-digital conversion errors and digital truncation effects become significant with respect to the magnitude of the scaled state variable values.

Secondly, normal scaling may be used, but each product and sum can be tested for overflow as it is formed, and if overflow is detected, then the computer can be programmed to substitute the maximum permissible value (± 5) for the intended product or sum. Quite obviously, if an optimal control scheme is being used, the trajectory will no longer be optimal if overflow occurs, but the steady state accuracy will not be affected once the state variables are finally driven to zero. This second approach will be developed herein since this method must be incorporated into the CDC 160 computer program.

The effects of this overflow problem on the specific system being simulated may best be illustrated on the X_1 vs X_2 phase plane. The area of concern

on this phase plane is the area bounded by the maximum digital-analog conversion limits, i.e., a square area centered at the origin bounded by $|X_1| = 5$ and $|X_2| = 5$. Overflow boundries for each of the products may be computed from Equation 4.3. Thus

$$|X_1| \leq 5 \tag{4.5}$$

and

$$|X_2| \leq 5/1.5 = 3.33 \tag{4.6}$$

The summation overflow boundries may be computed from Equation 4.4, i.e.,

$$|-X_1 - 1.5X_2| \leq 5$$

or

$$-.667X_1 - 3.33 \leq X_2 \leq -.667X_1 + 3.33 \tag{4.7}$$

The six straight lines defined by the equality relationships of Equations 4.5, 4.6 and 4.7 reduce the original digital-analog conversion area of the X_1 vs X_2 phase plane as shown in Figure 4-3. In this figure, Region I represents the product overflow area; Region II the summation overflow area; and Region III represents the area where both product and summation overflow occurs. Any initial condition or state variable sample value which lies in any of these regions will cause an overflow condition with resultant errors in computation. It should be noted that these overflow boundries are determined directly from the values of the state variable feedback coefficients, a_i. If these coefficients are changed, the overflow boundries also change according to Equations 4.3 and 4.4.

The dynamic behavior of the system described by Equation 4.1 can be expressed in terms of the discrete difference equation

$$\underline{X}(k+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \underline{X}(k) + \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix} U(k) \tag{4.8}$$

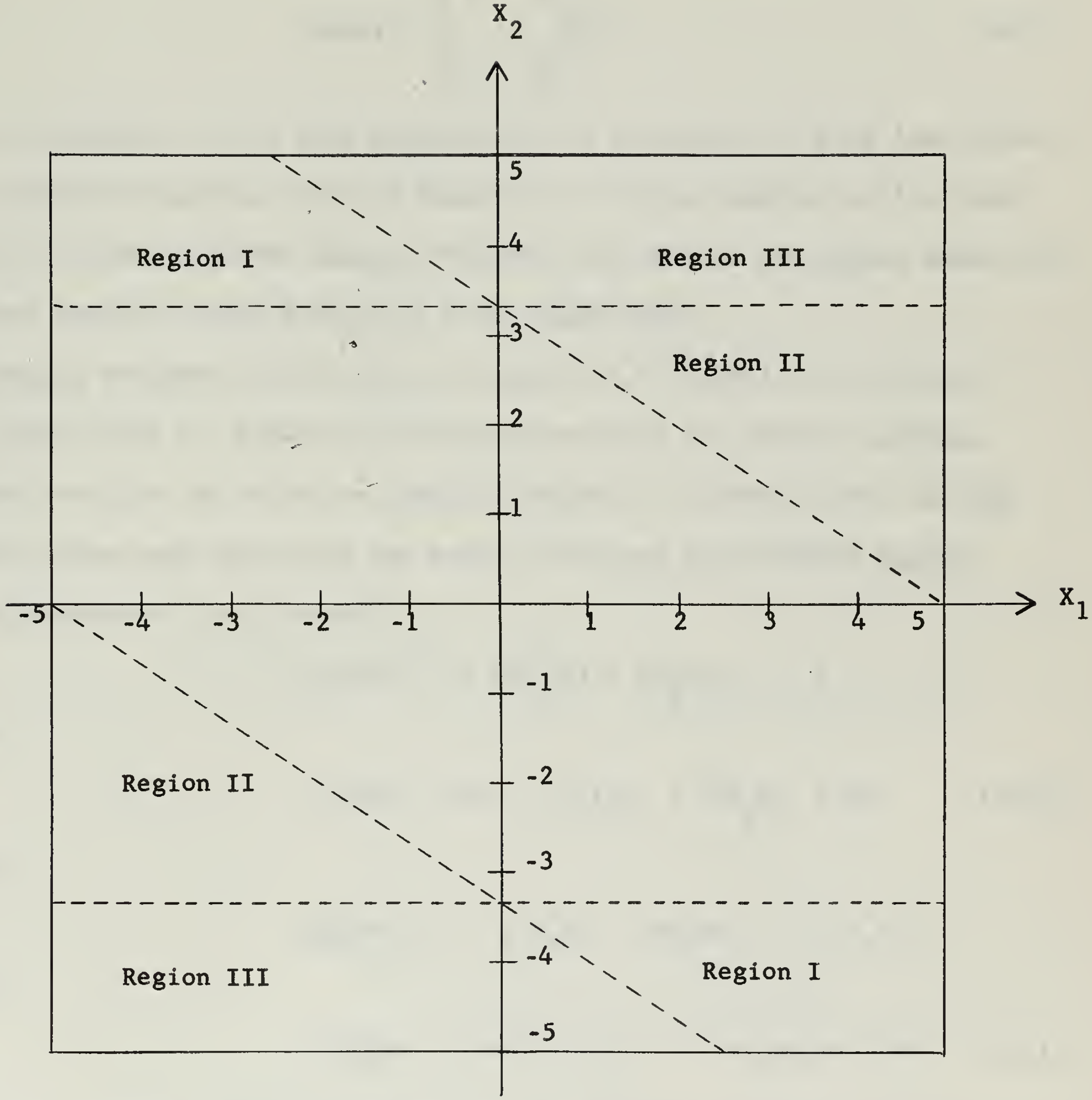


Figure 4-3. Phase Plane Overflow Regions

For a time-optimal trajectory, the control input must satisfy Equation 4.2. Combining Equations 4.2 and 4.8, the difference equation can be reduced to

$$\underline{X}(k+1) = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} \\ & \\ -1 & -\frac{1}{2} \end{bmatrix} \underline{X}(k) \tag{4.9}$$

It is important to note that Equation 4.9 is valid only if $\underline{X}(k)$ lies within the overflow boundries shown in Figure 4-3. If the state point lies outside of the non-overflow region, U(k) will not take on the proper value for optimal control, hence Equation 4.9 no longer holds.

Keeping the above limitations on Equation 4.9 in mind, not only must the state point lie within the overflow boundries for the kth sampling instant but also at the k+1st sampling instant. In other words, the kth control input must not drive the state point into the overflow region. From Equation 4.5, 4.6 and 4.9.

$$|X_1(k+1)| = |\tfrac{1}{2}X_1(k) + \tfrac{1}{4}X_2(k)| \leq 5$$

or

$$-2X_1(k) - 20 \leq X_2(k) \leq -2X_1(k) + 20 \tag{4.10}$$

and

$$|X_2(k+1)| = |-X_1(k) - \tfrac{1}{2}X_2(k)| \leq 3.33$$

or

$$-2X_1(k) - 6.66 \leq X_2(k) \leq -2X_1(k) + 6.66 \tag{4.11}$$

The two limiting straight lines expressed in Equation 4.10 lie entirely outside the digital-analog conversion boundry and impose no restrictions; however, the two lines defined by Equation 4.11 further reduce the non-overflow area of the phase plane as shown in Figure 4-4. This figure shows the optimal control boundries of the phase plane; if the initial state point is located within this boundry, optimal control will be realized.

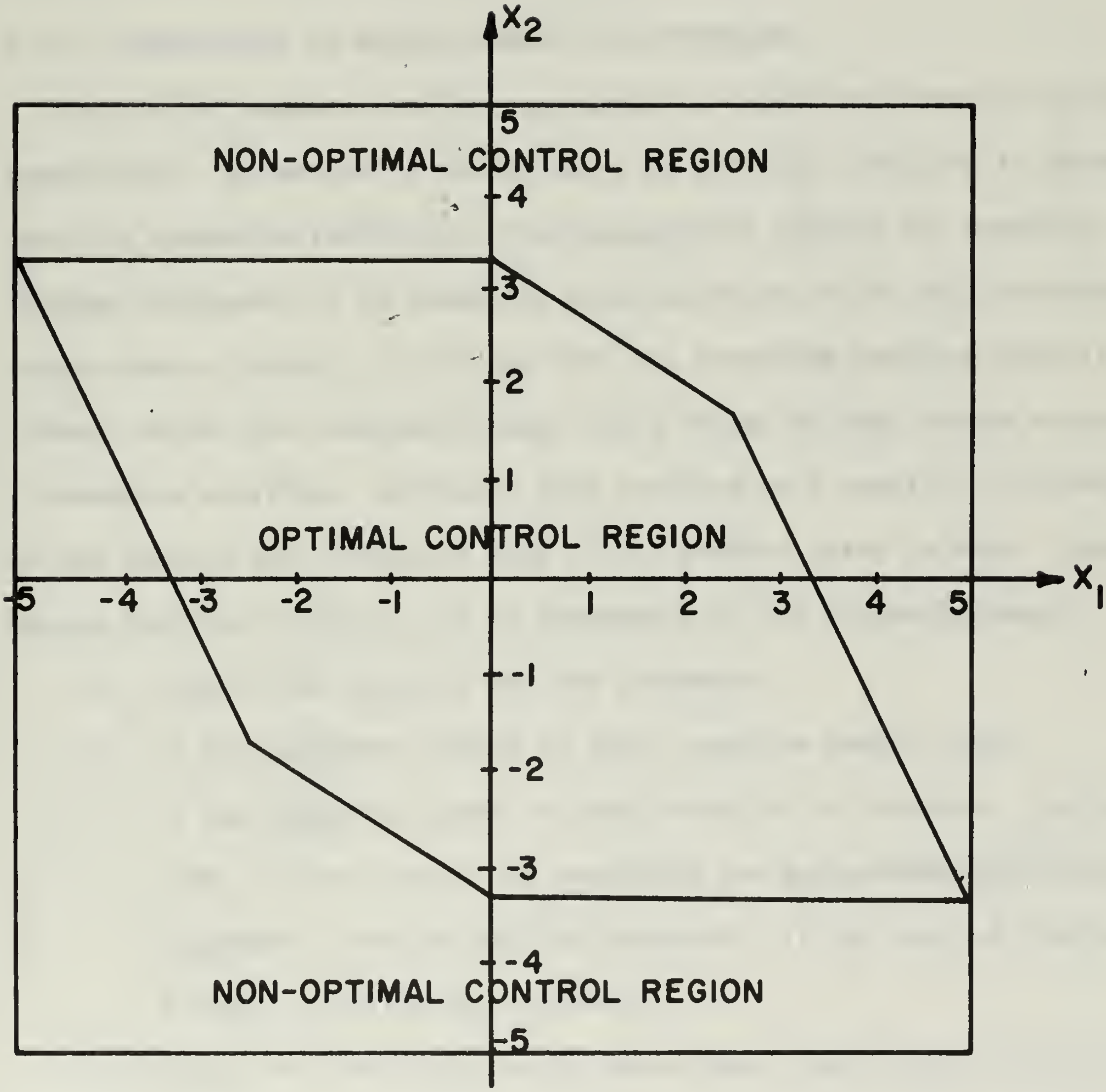


FIGURE 4-4 PHASE PLANE TIME-OPTIMAL CONTROL REGION

It should be emphasized that the state point may be driven to the origin from any point within the digital-analog conversion boundry provided that the CDC 160 computer program is written to correct for overflow; however, the trajectory will be time-optimal only if the initial state point lies within the boundries shown in Figure 4-4.

4.1.2 PROGRAMMING TO DETECT/CORRECT FOR OVERFLOW.

The CDC 160 computer can be programmed to detect and correct overflow conditions. The method of detection of an overflow condition is dependent upon the operation (addition or multiplication) causing the overflow. Referring to Figure 4-5 (a schematic representation of the digital-analog psudeo-number system), it is seen that the summation overflow path is continuous across the overflow boundry and a change of sign always occurs for a summation overflow. Note also that overflow as a result of the addition of two numbers only occurs if both of the numbers agree in sign. Thus summation overflow detection can be programmed in the following manner.

1. Compare the signs of the two arguments.
2. If the arguments differ in sign, overflow cannot occur.
3. If the arguments agree is sign, overflow is possible. In this case, if the sign of the resultant sum agrees with the sign of the arguments, overflow has not occurred. If the sign of the sum differs, overflow has occurred.

The product overflow path, on the other hand, cannot cross the overflow boundary as can be seen in Figure 4-5. The overflow product does not change sign but undergoes a discontinuous jump in magnitude as overflow occurs. Product overflow can only occur if both of the arguments are greater than unity. Thus to detect overflow of the product $a_i X_i$:

1. Determine if a_i is less than unity. If so, overflow cannot occur and no further tests are required.

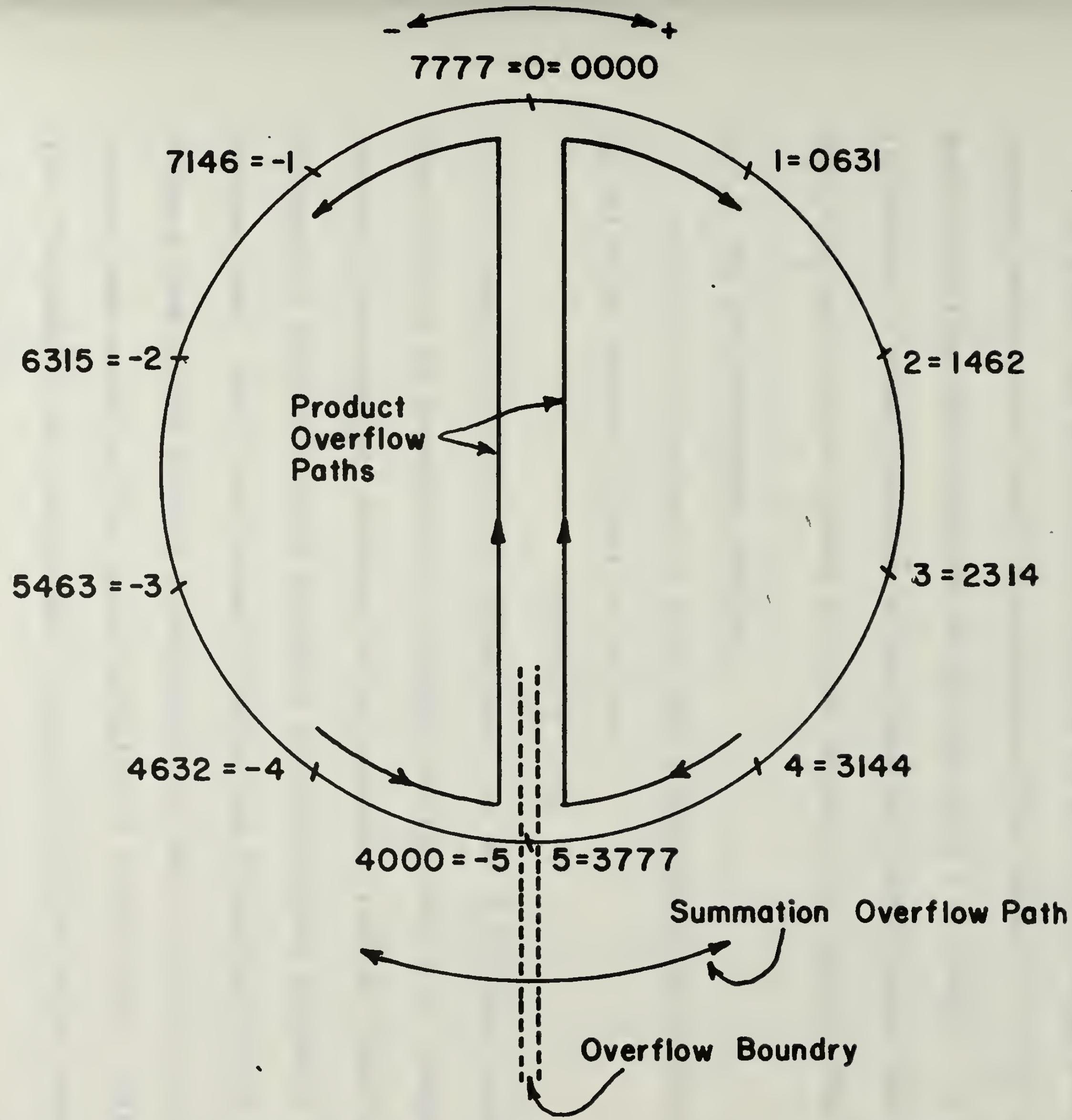


FIGURE 4-5 ANALOG-DIGITAL NUMBER SYSTEM AND OVERFLOW PATHS

2. If a_i is greater than unity, compare the magnitudes of $a_i X_i$ and X_i. If the magnitude of the product is less than the magnitude of X_i itself, overflow has occurred; otherwise not.

Overflow correction in both cases consists of substituting the maximum possible value ($\pm$ 5) for the overflow quantity consistent with the signs of the arguments causing overflow.

To detect and correct for overflow in the system being simulated, a product overflow test must be performed on the product $(-1.5X_2)$ and a summation overflow test performed on the sum $(-X_1) + (-1.5X_2)$. If the program is written to allow variation of the feedback coefficients, then the product a_1X_1 must also be tested to allow values of a_1 greater than unity.

4.2.0 THE NON-AUTONOMOUS PROBLEM.

Suppose it is desired to force the system described by Equation 4.1 to follow an external signal of the form

$$R(t) = r_o + \dot{r}t \tag{4.12}$$

If one associates $X_1(t)$ with system position and $X_2(t)$ with system velocity, then R(t) can be considered as "target" position and $\dot{r}$ as "target velocity. Bertram's time-optimal solution of the regulator problem (Equation 4.2) is designed to drive the state variables of the original system to zero in minimum time. This same optimal solution can be applied to the non-autoonomous case if one defines a position error and a velocity error and considers these errors as the state variables of a new system. With the inputs thus "imbedded", the time-optimal solution will drive these errors to zero in minimum time.

Define:

$$\begin{aligned} \text{position error} &= Y_1(t) = X_1(t) - R(t) \\ \text{velocity error} &= Y_2(t) = X_2(t) - \dot{r} \end{aligned} \tag{4.13}$$

Applying Bertram's time-optimal forcing function, U, to this new system:

$$U(k) = -Y_1(k) - 1.5Y_2(k)$$

which by substitution of Equations 4.12 and 4.13 becomes

$$U(k) = -[X_1(k) - r_o - \dot{r}t(k)] - 1.5[X_2(k) - \dot{r}] \qquad (4.14)$$

The simulation of this system can be accomplished in the same manner as the regulator problem adding the requirement that the CDC 160 computer take into account target position and velocity in addition to system position and velocity. The simulation scheme chosen is shown in Figure 4-6. The value r_o is set into the CDC 160 computer manually due to the four-channel limitation on the ADC. Note that in an actual system, it is possible for the signal R(t) to be transmitted to the digital computer from an entirely remote source such as a radar system.

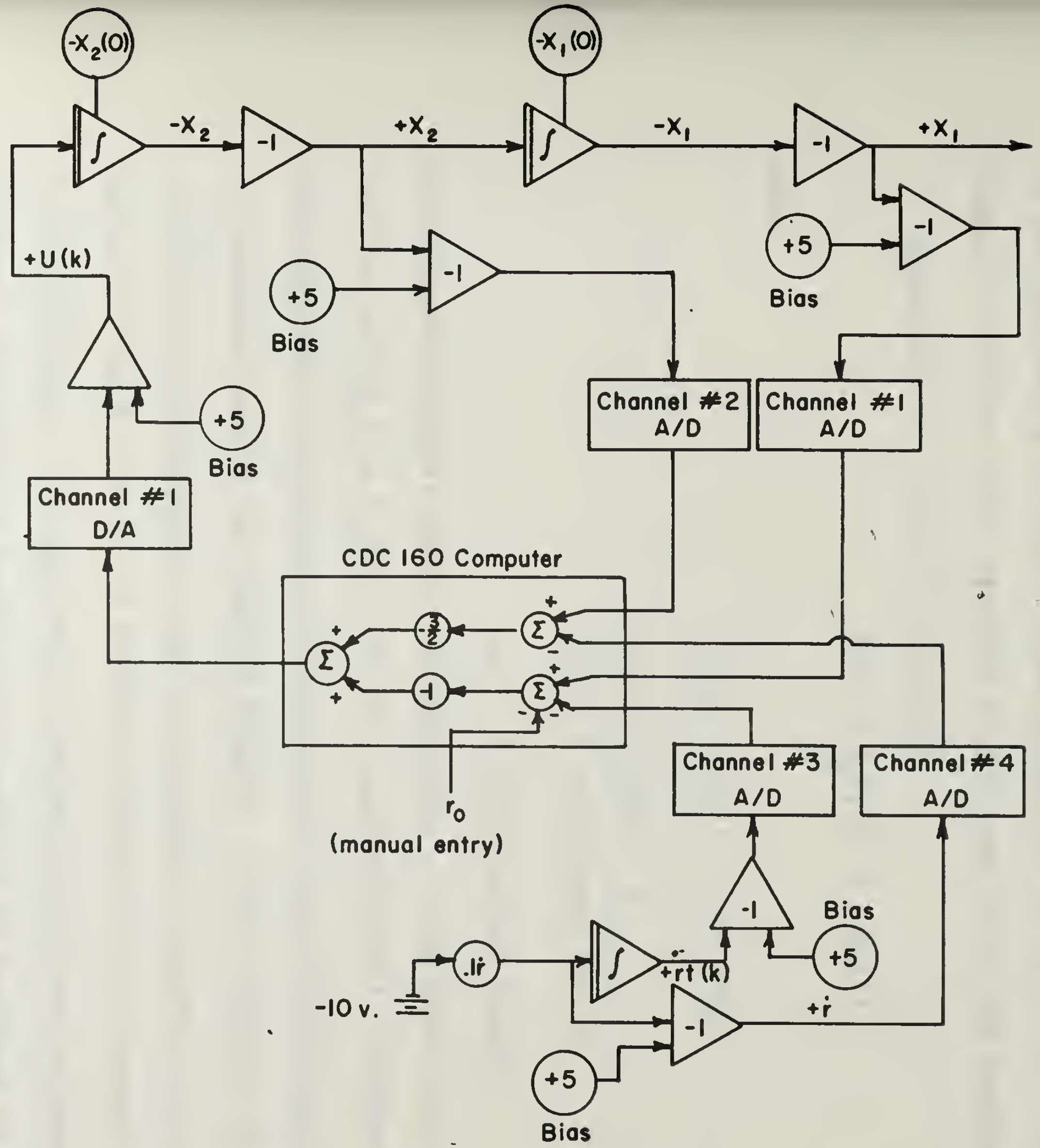


FIGURE 4-6 SIMULATION SCHEME FOR ZERO AND NON-ZERO INPUTS

4.2.1 LIMITATIONS IMPOSED BY HYBRIDIZATION.

As with the regulator problem, the most severe limitations imposed on the non-autonomous system are due to overflow phenomena which impose maximum limits on the signal R if optimum control is to be realized. In the computation of Equation 4.14, the 160 computer must perform four additions and two multiplications. At first glance, it appears that a total of six overflow tests must be performed; one for each operation involved in the computation of U(k). Closer examination will show that the actual number of overflow tests required is the same as for the regulator problem.

Equation 4.14 can be written as:

$$U(k) = a_1 S_1(k) + a_2 S_2(k) \tag{4.15}$$

where

$$S_1(k) = X_1(k) - r_o - \dot{r}t(k)$$

$$S_2(k) = X_2(k) - \dot{r}$$

$$a_1 = -1 \text{ for optimal control}$$

$$a_2 = -1.5 \text{ for optimal control.}$$

The 160 computer must form S_1 in two steps. If the first step is the addition of $(-r_o)$ to $X_1(k)$, then this sum may possibly overflow. If overflow does occur, reference to Figure 4-5 will show that summation overflow is continuous across the overflow boundry. Further addition of the quantity $(-\dot{r}t(k))$, if of appropriate sign and magnitude, may thus be able to bring the entire sum S_1 out of the overflow condition. If it cannot, then S_1 itself overflows. The same result obtains if the order of addition is reversed. Therefore an overflow test need not be performed on the intermediate sum in the computation of S_1 and only one overflow test is required on the entire composite sum.

Noting that for a one second sampling interval, $t(k) = k$, then Equations 4.14 and 4.8 may be combined to yield:

$$\underline{X}(k+1) = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} \\ -1 & -\frac{1}{2} \end{bmatrix} \underline{X}(k) + \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix} [r_o + \dot{r}\{k + 1.5\}] \tag{4.16}$$

Assume $\underline{X}(0) = 0$. Then from Equation 4.16,

$$\underline{X}(1) = \begin{bmatrix} .5r_o + .75\dot{r} \\ r_o + 1.5\dot{r} \end{bmatrix} \tag{4.17}$$

$$\underline{X}(k) = \begin{bmatrix} r_o + k\dot{r} \\ \dot{r} \end{bmatrix}, \quad k = 2, 3, \ldots \tag{4.18}$$

To prevent overflow of the sum S_1:

$$|S_1(k)| = |X_1(k) - r_o - \dot{r}k| \leq 5 \tag{4.19}$$

Evaluating this inequality for various values of k by the substitution of Equations 4.17 and 4.18 (still under the assumption that $\underline{X}(0) = 0$), one obtains:

(1) $|r_o| \leq 5$ for $k = 0$ (4.20)

(2) $-20 - 2r_o \leq \dot{r} \leq 20 - 2r_o$ for $k = 1$. (4.21)

(3) $|S_1(k)| = 0$ for $k \geq 2$.

Equations 4.20 and 4.21 define the overflow boundries of S_1 as a function of r_o and $\dot{r}$. S_1 can only overflow for $k = 0$ or $k = 1$. (Note that these same limitations on r_o and $\dot{r}$ also hold for the quantity $a_1S_1(k)$ if $a_1 = -1$).

Similar overflow analysis can be performed on the quantities $S_2(k)$, $a_2S_2(k)$ and $U(k)$. Table 4-1 lists the results of this overflow analysis. If the straight lines defined by the equality relationships of each of the input limitations of Table 4-1 are plotted on the r_o vs $\dot{r}$ plane, the overflow boundries for the system are defined. The most severe limitations are

determined by the quantities a_2S_2 and $U(k)$, and thus determine the range of inputs allowable for optimal control. These limiting boundries are shown by the solid lines of Figure 4-7. It should be noted that Table 4-1 and Figure 4-7 are valid only under the assumption that $\underline{X}(0) = 0$. For this special case, the optimal control region under non-zero input conditions is equivalent to the optimal control region on the X_1 vs X_2 phase plane for the regulator problem. (See Figure 4-4).

QUANTITY	INPUT LIMITATIONS		
	$k = 0$	$k = 1$	$k \geq 2$
$S_1(k)$ & $a_1 S_1(k)$ (if $a_1 = -1$)	$\lvert r_o \rvert \leq 5$	$\dot{r} \geq -20 - 2r_o$ $\dot{r} \leq 20 - 2r_o$	None
$S_2(k)$	$\lvert \dot{r} \rvert \leq 5$	$\dot{r} \geq -10 - 2r_o$ $\dot{r} \leq 10 - 2r_o$	None
$a_2 S_2(k)$	$\lvert \dot{r} \rvert \leq 3.33$	$\dot{r} \leq 6.67 - 2r_o$ $\dot{r} \geq -6.67 - 2r_o$	None
$U(k)$	$\dot{r} \leq 3.33 - .667r_o$ $\dot{r} \geq -3.33 - .667r_o$	$\dot{r} \leq 10 - 2r_o$ $\dot{r} \geq -10 - 2r_o$	None

Table 4-1. Input Limitations for $\underline{X}(0) = 0$.

A further limitation on problem running time is imposed due to the fact that X_1 will eventually exceed the five volt A/D conversion limit as it follows target position. This limitation can be expressed as:

$$|\dot{r}| \leq \frac{5 - r_o}{k}, \quad \underline{X}(0) = 0, \; k = 1, 2, 3, \ldots \tag{4.22}$$

The limiting case of this relation for various values of k is shown by the dotted lines of Figure 4-7. If a problem running time of ten seconds (k = 10) is desired, for example, then $|\dot{r}| \leq \frac{1}{2} - r_o/10$. In a practical situation, A/D conversion equipment must be chosen so as to provide a conversion limit which exceeds the maximum expected value to be converted, or alternatively, successive scale factors must be applied to the quantity X_1 as the conversion limit is approached. This limitation cannot be removed by appropriate programming of the digital computer.

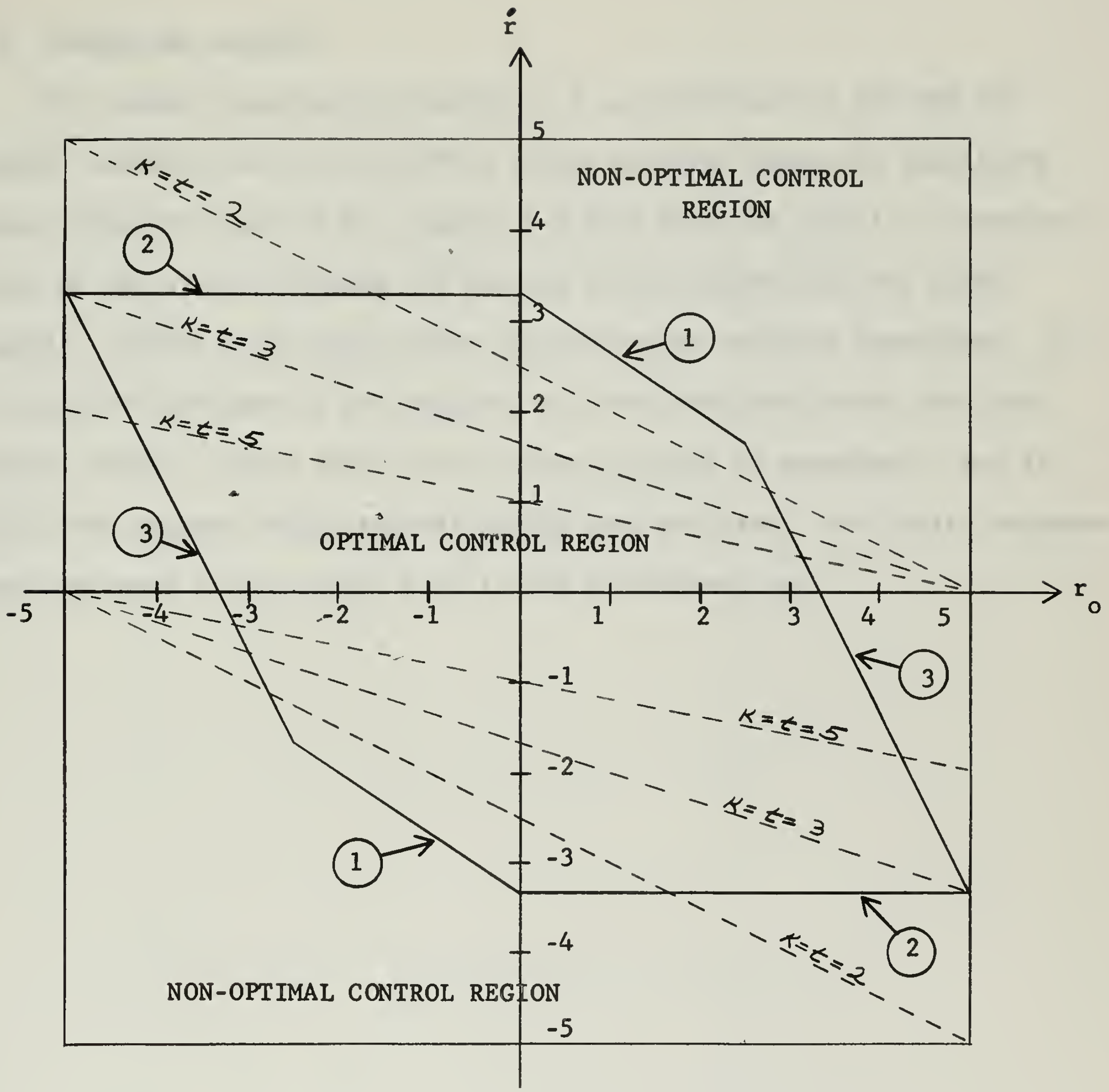
As a result of the above overflow analysis, and if one programs the digital computer to permit variation of the feedback coefficients a_1 and a_2 (and thereby varying the overflow boundries), the following quantities must be tested and corrected for overflow:

(1) The product $a_1S_1(k) = a_1 \, [X_1(k) - r_o - \dot{r}t(k)]$.

(2) The product $a_2S_2(k) = a_2[X_2(k) - \dot{r}]$.

(3) The sum $U(k) = a_1S_1(k) + a_2S_2(k)$.

If r_o and $\dot{r}$ are zero, then the above equations reduce to the overflow quantities considered in the regulator problem discussed in Section 4.1. Thus one computer program will permit control of the system under both zero and non-zero input conditions. Program DIGIT (Appendix IV) is a digital computer program written for the CDC 160 computer which performs the above overflow tests and corrects the overflow condition based on the simulation scheme shown in Figure 4-6.



KEY:

1 U(0) overflow boundry .

2 $-1.5S_2(0)$ overflow boundry.

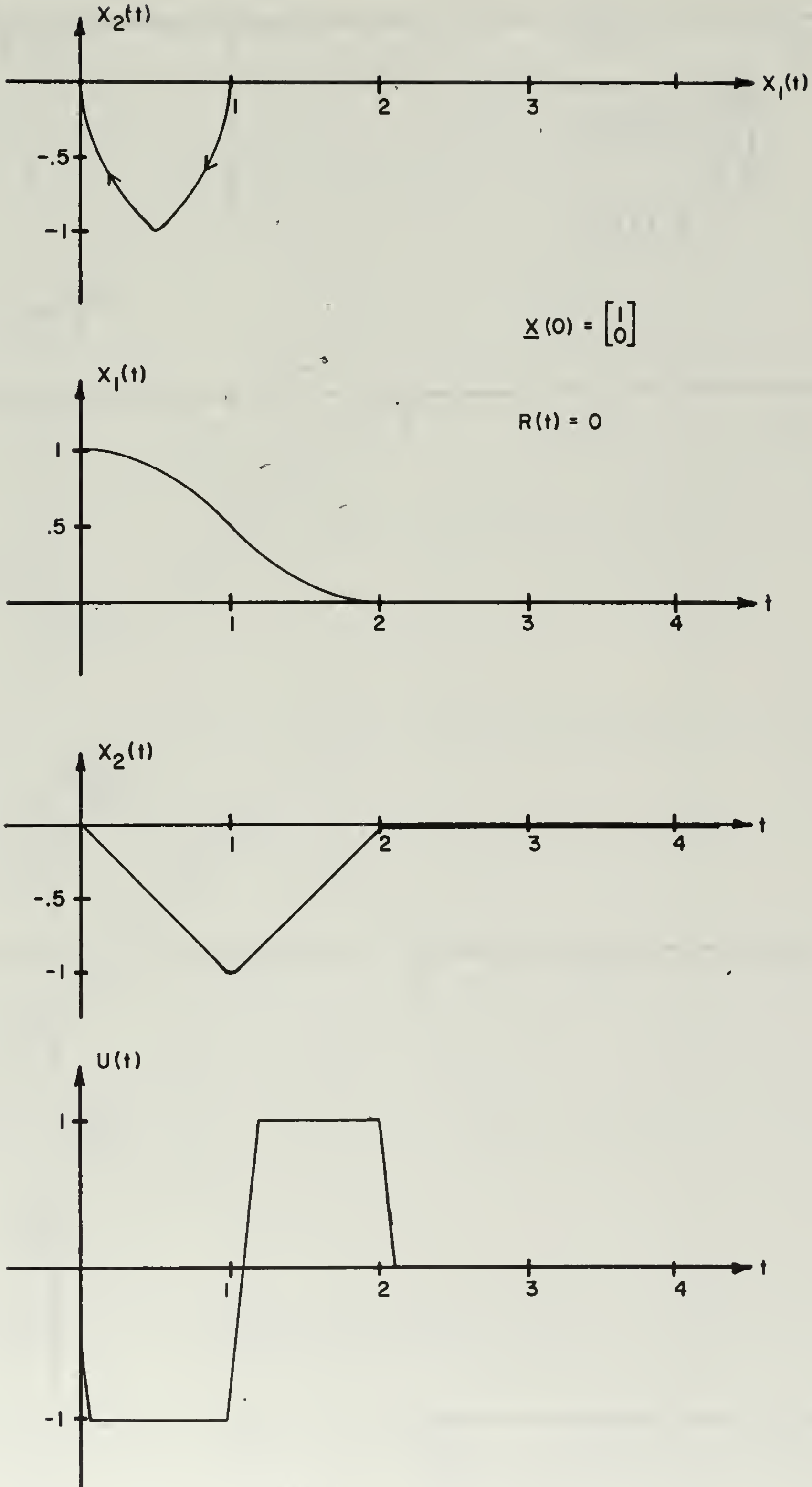
3 $-1.5S_2(1)$ overflow boudry.

- - - X_1 overflow at time indicated.

Figure 4-7. Input Limitations for $\underline{X}(0) = 0$.

4.3 SIMULATION RESULTS.

The system described by Equation 4.1 was simulated on the CDC 160 digital computer and the PACE TR-20 analog computer using the simulation scheme shown in Figure 4-6. Figures 4-8 thru 4-13 are actual x-y recorder plots of the system response for various initial conditions and input signals. Figure 4-10 shows system response under overflow conditions. It is seen that two samples are required to drive the state point into the optimal control region after which optimal control is exercised. For all plots, the program DIGIT internal timing loop was used. The finite response time indicated on the signal $\mathbf{U}(t)$ is due to recorder lag.



Simulation Results for Initial Position Error

Figure 4-8

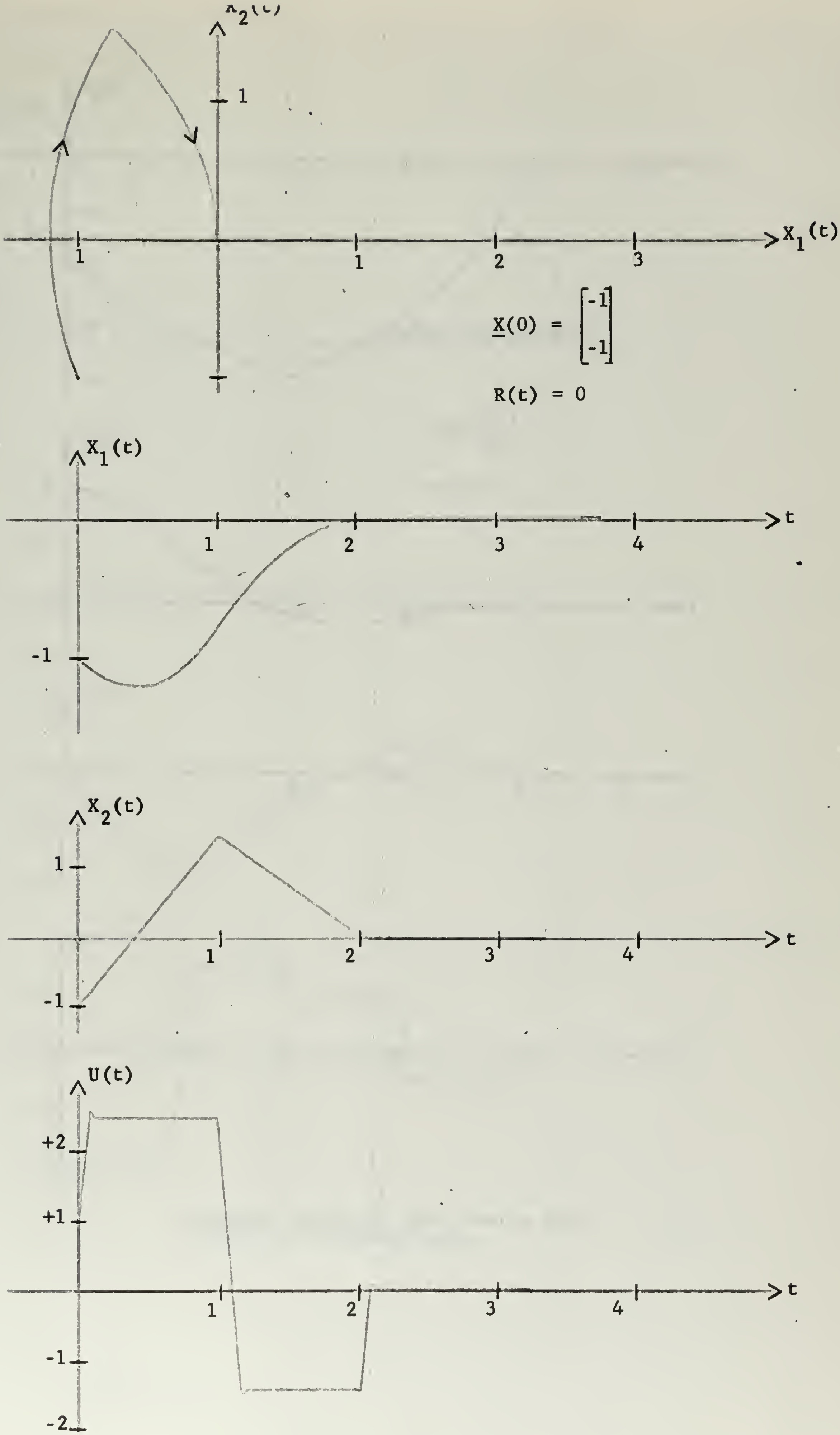
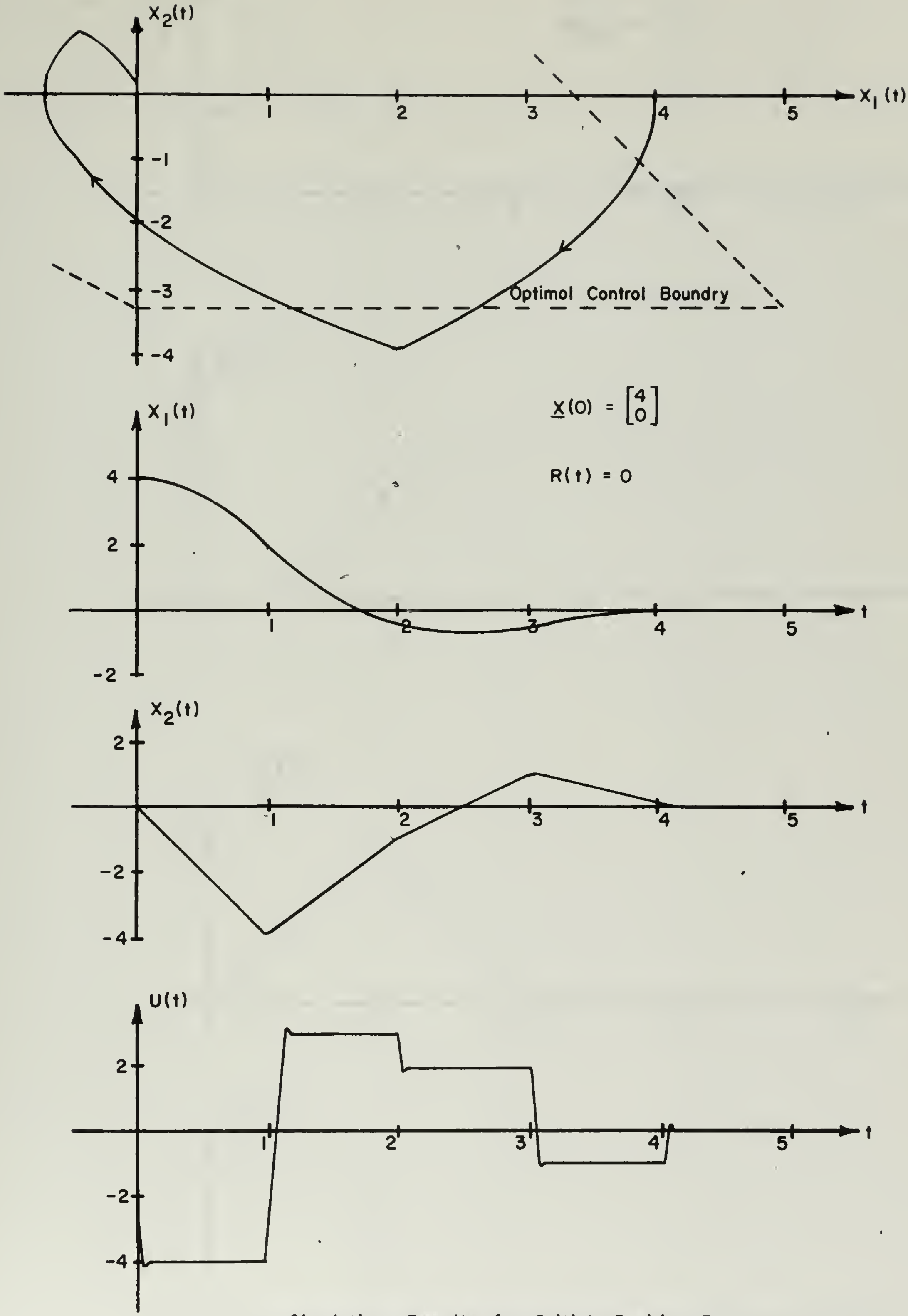
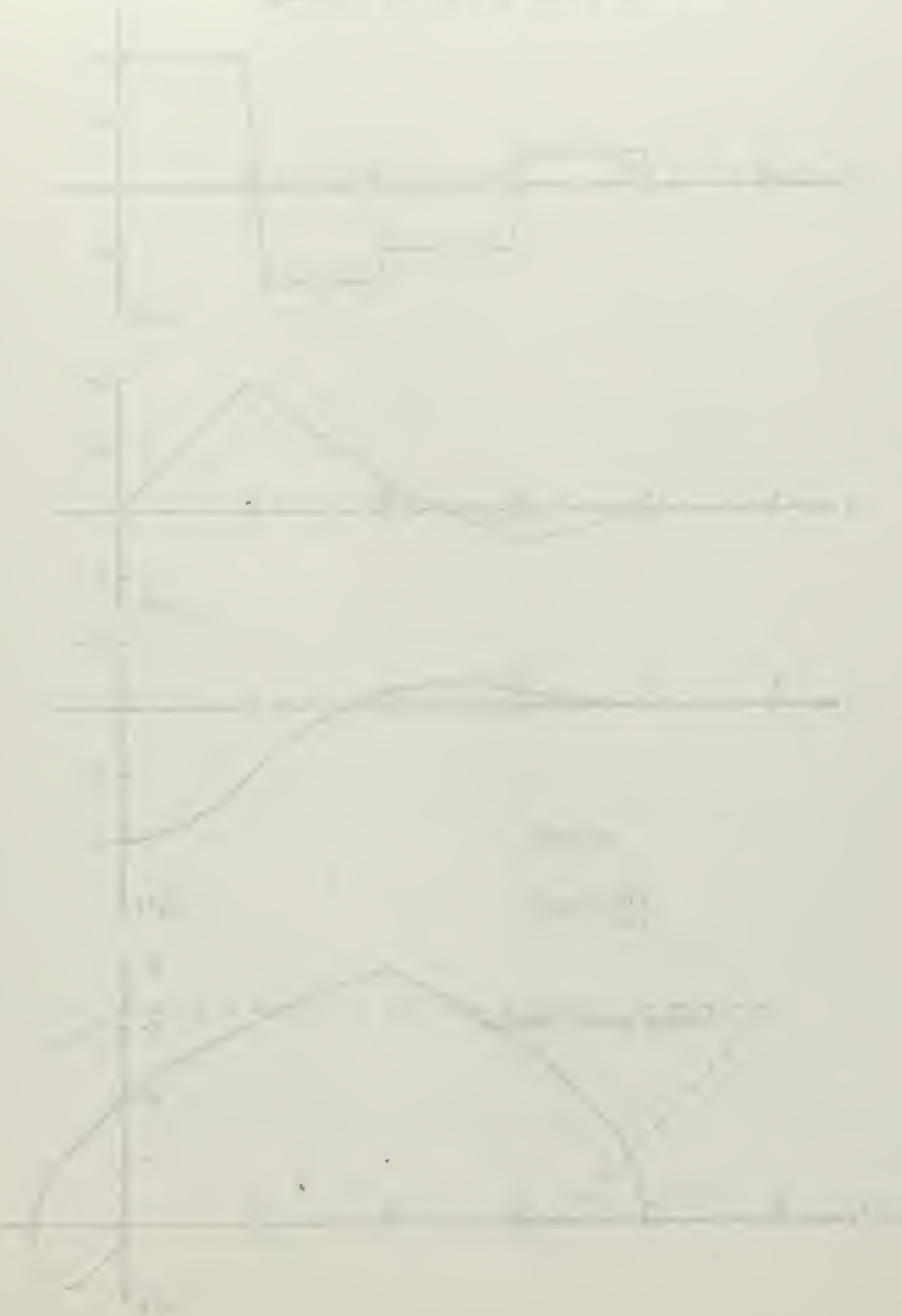


Figure 4-9 Simulation Results for Initial Position and Velocity Error



Simulation Results for Initial Position Error Resulting in Non-Optimal Control.

Figure 4-10



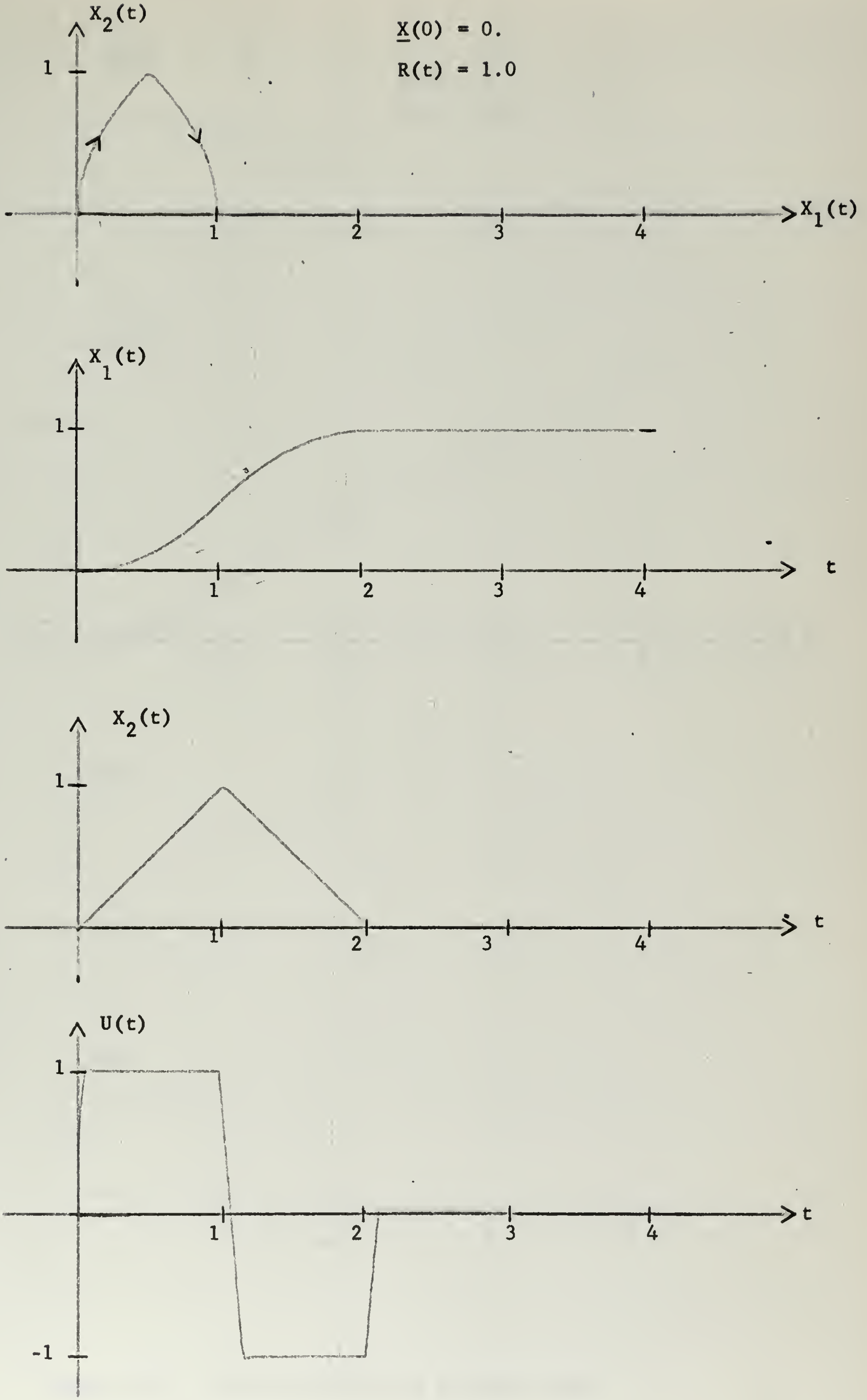


Figure 4-11. Simulation Results for Step Input

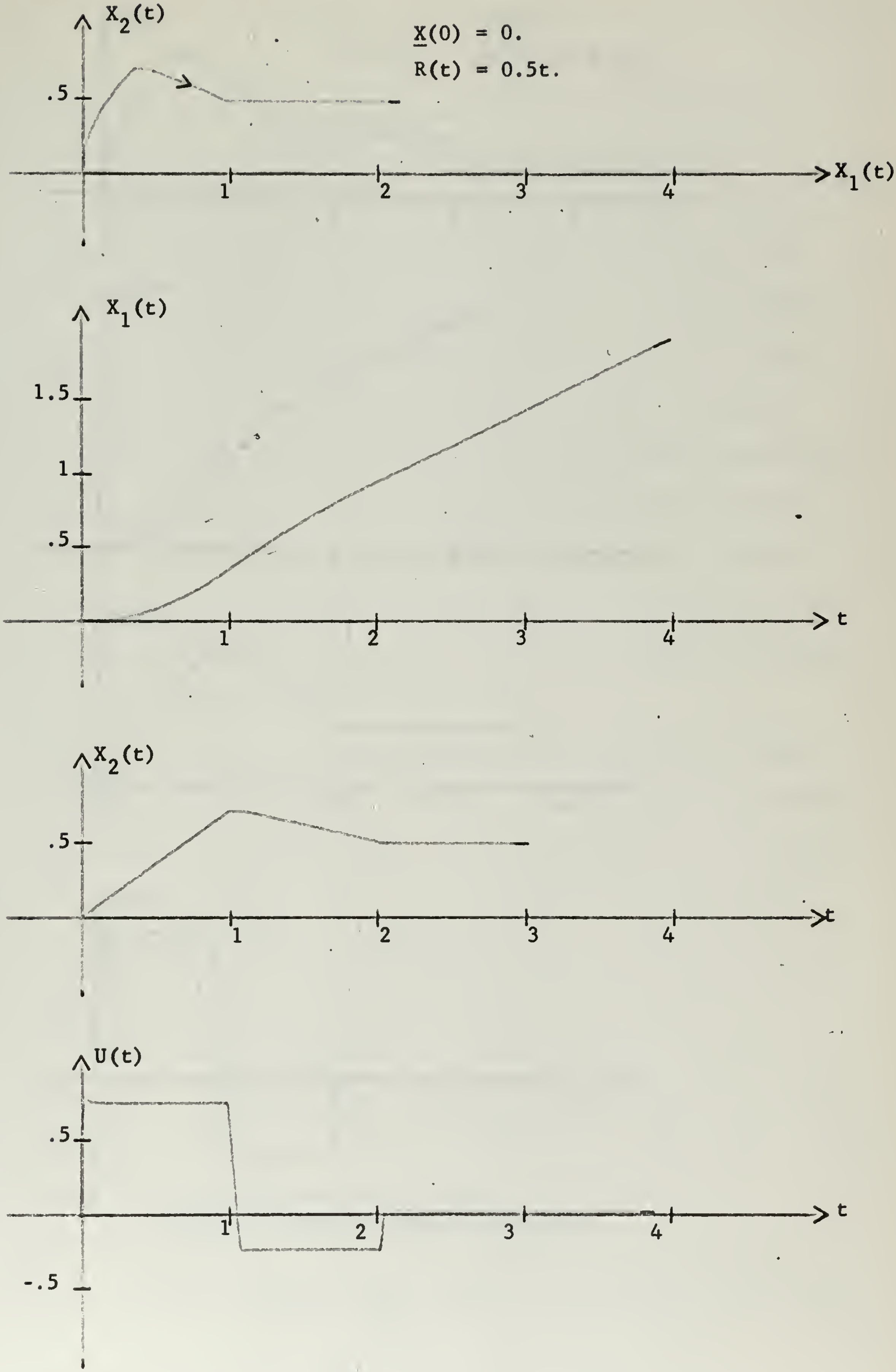
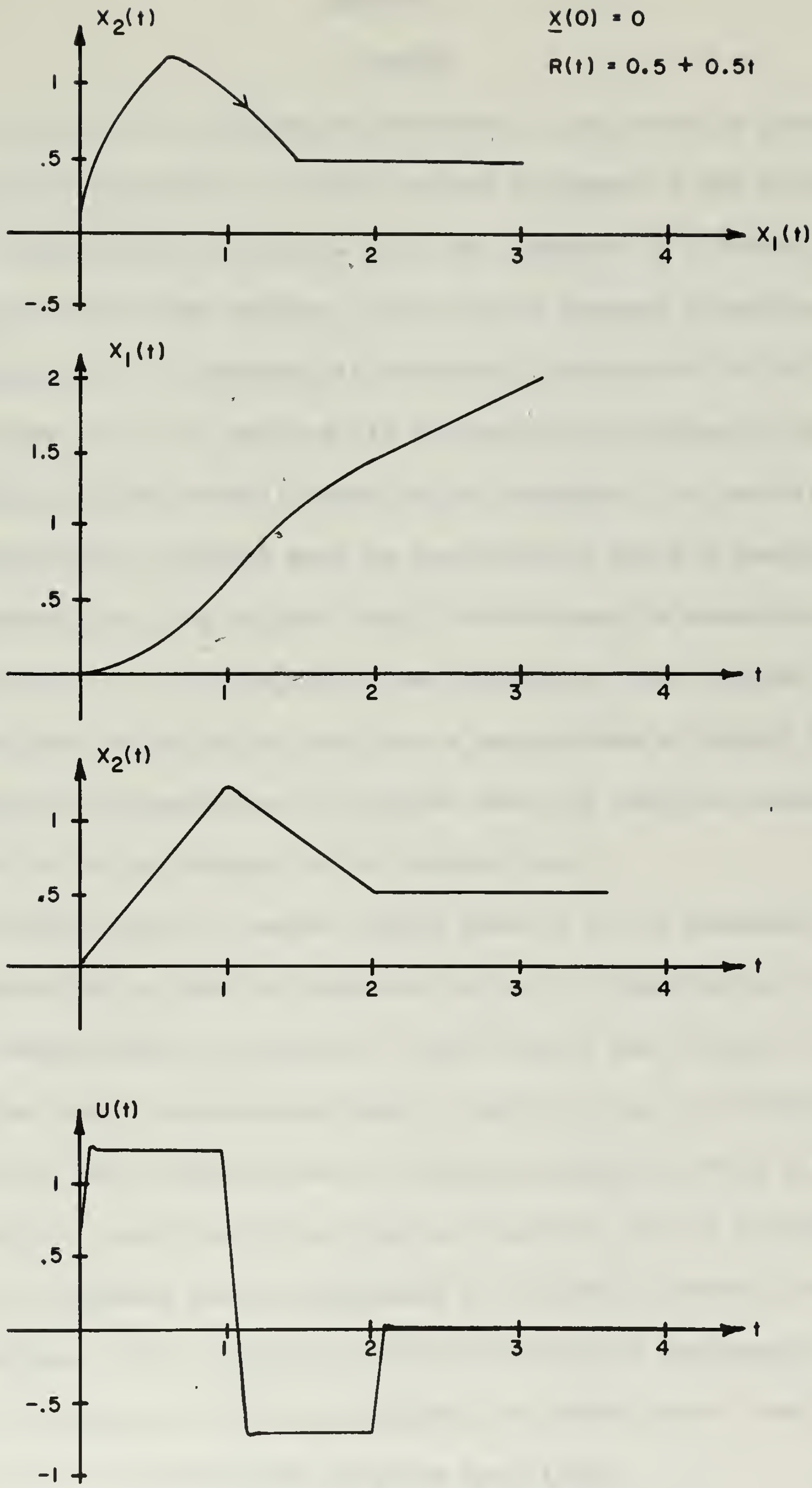


Figure 4-12. Simulation Results for Ramp Input



Simulation Results for Step plus Ramp Input.

Figure 4-13

CHAPTER 5

SUMMARY

One of the first problems encountered in the use of a short word length digital computer for hybrid control purposes is the problem of entry of computational constants into the computer in a format compatible with the A/D conversion process. Two choices present themselves at this point; namely, (1) to perform all arithmetic operations in the A/D psuedo-number system or (2) to perform all arithmetic operations in the standard binary number system normally used by the computer. In the first case, the computational constants must be converted to the A/D number system and in the second case, the digital sample values must be converted to normal binary notation. The first choice was chosen for this project resulting in subroutine KMOD which can be used for a large class of hybrid control programs wherein constants must be entered into the computer manually and must be capable of being changed by the program user.

The above choice of number system results in the phenomena of number system "overflow" within the computer during the computation of the control law. As demonstrated in Chapter 4, each term of the control law is capable of overflow under certain conditions. The two types of overflow (product or summation) are characterized by certain properties which allow detection of an overflow condition by the digital computer, and if overflow is detected, the computer can be programmed to partially correct for this overflow condition. This overflow detection/correction programming can greatly increase the range of initial conditions or system inputs even though optimal control is not realized under overflow conditions.

Program DIGIT, written for the specific system simulated in Chapter 4, is sufficiently general so that it may be used with any second-order analog

simulation where the control law to be computed by the digital computer consists of a linear combination of the state variable values. A wide range of feedback coefficients and variable sample timing can be accommodated by this program.

Further investigation is needed on the second method of arithmetic computation, i.e., the conversion of the A/D digital samples to normal binary notation. It is believed that this method is most easily implemented on a long word length computer possessing the capability of floating point arithmetic. The programming difficulties presented by this conversion process appear to be excessive for a short word length, fixed point arithmetic machine such as the CDC 160 computer, and since this conversion must be included in the sample loop of the program, excessive computation time between the input of samples and the output of the control value may result.

BIBLIOGRAPHY

1. Bertram, J. E. The Concept of State in the Analysis of Discrete-Time Control Systems. Workshop on State Space Techniques for Control Systems, 1962.

APPENDIX I

OPERATION OF THE CDC 168 ARITHMETIC UNIT

I-1. MULTIPLICATION AND DIVISION OPERATIONS.

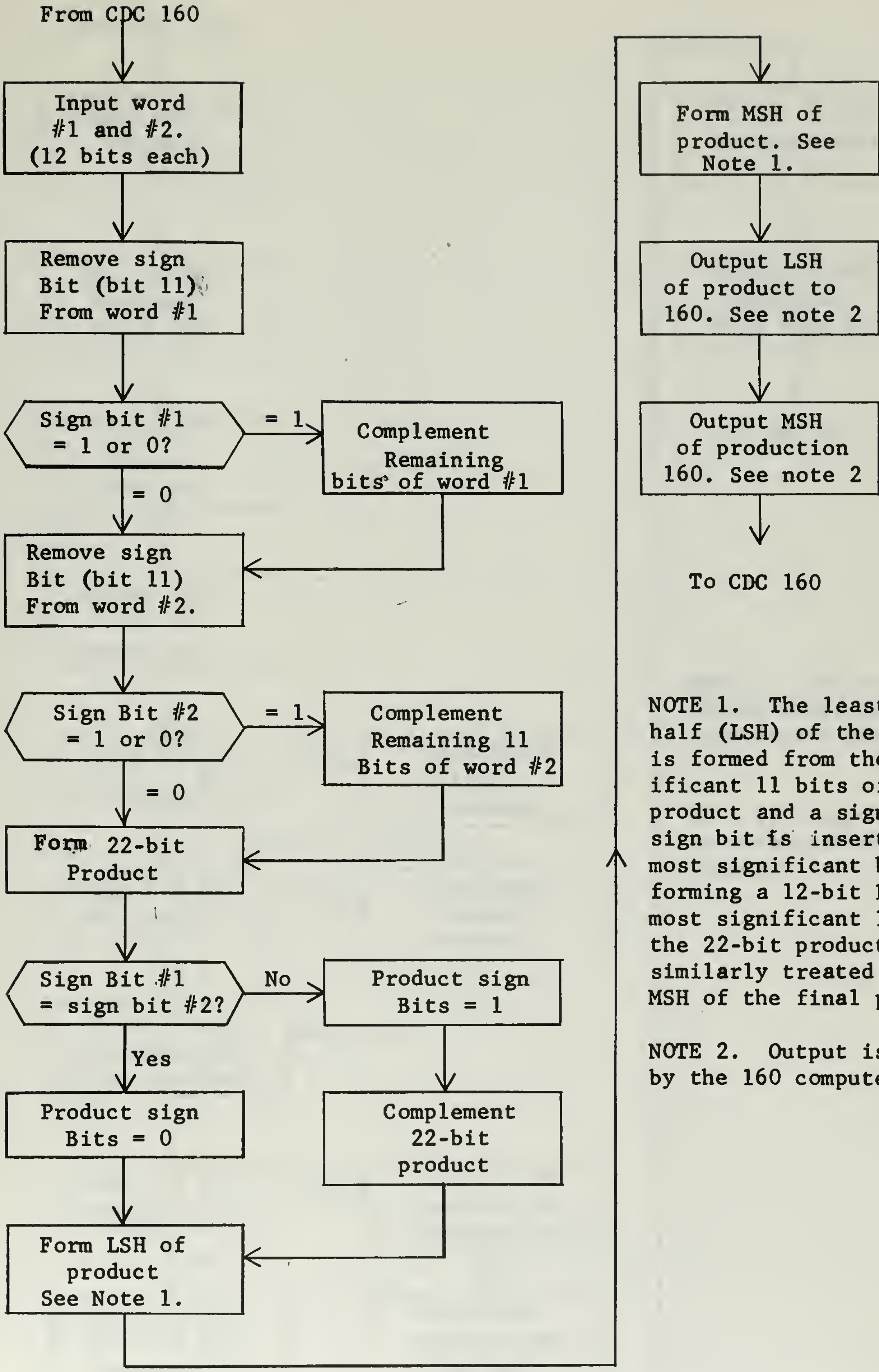
The CDC 168 arithmetic unit is used in conjunction with the CDC 160 computer for single precision (short) multiplication and division in the program developed herein. A detailed discussion of these two modes of operation is given below.

A. Single Precision (Short) Multiply. To initiate the short multiply operation, the 160 computer selects the arithmetic unit and transmits a 12-bit multiplicand and a 12-bit multiplier to the arithmetic unit. The steps by which the arithmetic unit computes the product are shown in Figure I-1. Upon completion of the short multiply operation, the product is located in two successive storage locations in the 160 computer; the least significant half (LSH) in the first storage cell, the most significant half (MSH) in the second cell.

B. Single Precision (Short) Divide. To initiate the short divide operation, the CDC 160 computer selects the arithmetic unit and transmits two 12-bit words which form the dividend followed by a 12-bit divisor. The two 12-bit words forming the dividend <u>must</u> agree in sign. Figure I-2 shows the step by step operation of the arithmetic unit in the short divide mode. It is to be noted that the 160 computer transmits the least significant half of the two word dividend first, and inputs the 12-bit quotient followed by the 12-bit remainder.

I-2. SHORT MULTIPLICATION AND DIVISION EXAMPLES.

As noted above, the short multiply operation yields a two word product in the 160 computer. Successive operations in the 160 computer, and digital-to-analog conversion are single word (12-bit) operations. Thus a



NOTE 1. The least significant half (LSH) of the final product is formed from the least significant 11 bits of the 22-bit product and a sign bit. The sign bit is inserted as the most significant bit, thus forming a 12-bit LSH. The most significant 11 bits of the 22-bit product are similarly treated to form the MSH of the final product.

NOTE 2. Output is initiated by the 160 computer.

Figure I-1. CDC 168 Arithmetic Unit Short Multiply Flow Chart

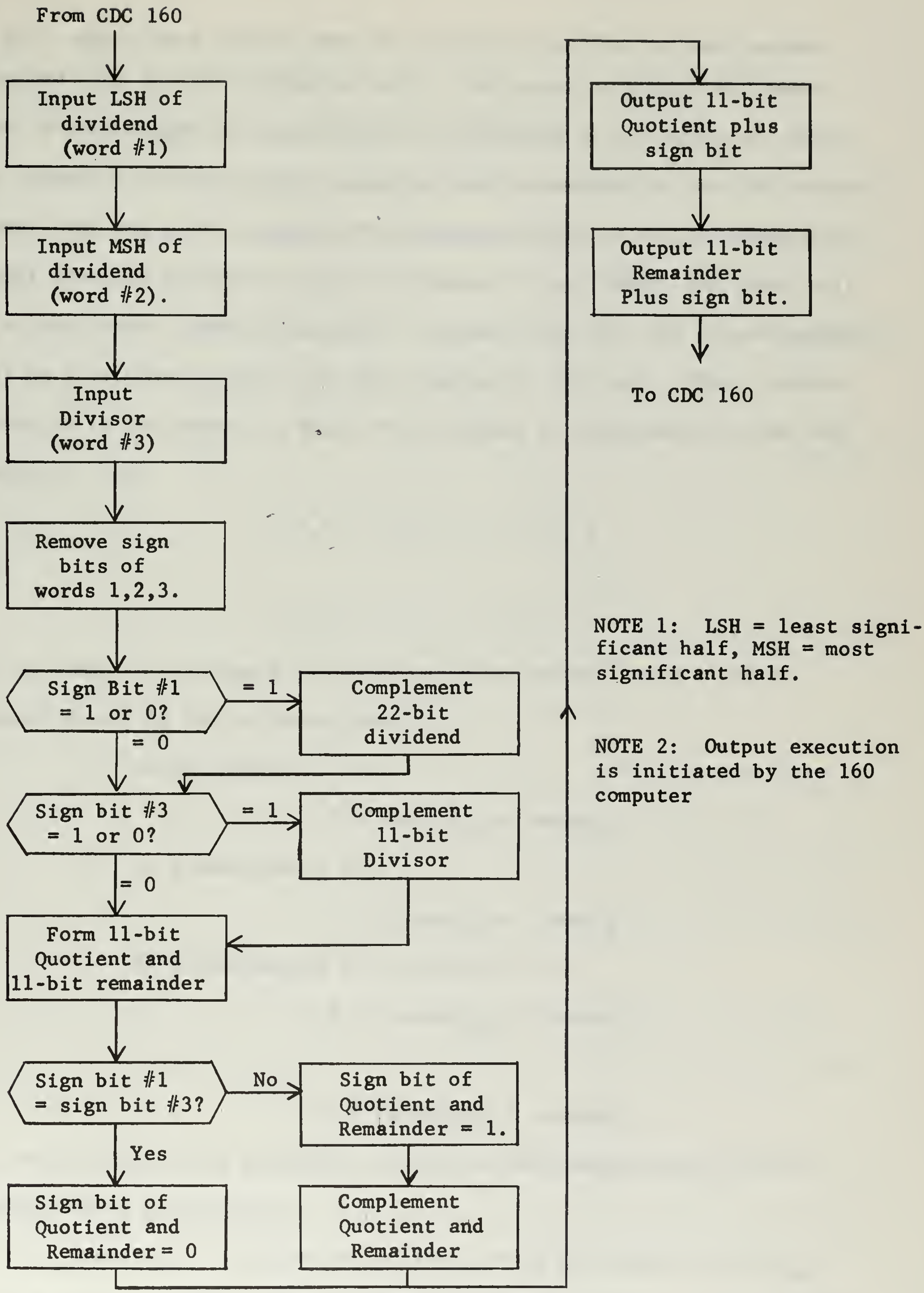


Figure I-2. CDC 168 Arithmetic Unit Short Divide Flow Chart.

12-bit, single word product must be retrieved from the two word answer received from the 168 arithmetic unit. The manner in which this reduction in word length is accomplished is determined by the manner in which the arguments of the multiply operation are represented in the 160 computer. Assume that one of the arguments (either multiplier or multiplicand) will result from the analog-to-digital conversion of a voltage, and hence will be of the format shown in Table 2-1. Assume also, that the second argument will be a constant entered into the computer by the user. Then a choice exists as to the manner in which this constant is represented in the 160 computer. Let

$$n = 0, 1, 2, \ldots, 7, 8 \text{ or } 9$$

and

$$m = 0, 1, 2, \ldots, 6 \text{ or } 7.$$

If the symbol K represents the constant to be entered, then K can be entered in one of the following ways:

1. As an integer, i.e.,

$$K = (nn\ldots)_{10} = (mmmm)_8$$

2. As a fraction, i.e.,

$$K = (0.nnn)_{10} = (.mmmm)_8$$

3. As a combination of the above, i.e.,

$$K = (n.nnnn)_{10} = (m.mmm)_8$$

or

$$K = (nn.nnn)_{10} = (mm.mm)_8$$

The radix point is not physically present in the computer word, but is included above for clarity.

A. Example 1. Consider the product, (K) x (Y), where $K = (-2)_{10}$ and $Y = (+2)_{10} = (1462)_8$ after analog-to-digital conversion. Using method

(1) to enter K into the 160 computer, then $(+2)_{10} = (0002)_8$ and to represent $(-2)_{10}$, the 7's complement is used, i.e.,

$$K = (-2)_{10} = \overline{(0002)}_8 = (7775)_8.$$

Referring to Figure I-1, the 168 arithmetic unit will form the product (K) x (Y) as follows:

$[(7775)_8 - \text{sign bit}] = (11\ 111\ 111\ 101)_2$ (sign bit = 1)

$\overline{[(7775)_8 - \text{sign bit}]} = (00\ 000\ 000\ 010)_2$

$[(1462)_8 - \text{sign bit}] = (01\ 100\ 110\ 010)_2$ (sign bit = 0)

$(\text{22-bit product}) = (0\ 000\ 000\ 000\ 011\ 001\ 100\ 100)_2$

$\overline{(\text{22-bit product})} = (1\ 111\ 111\ 111\ 100\ 110\ 011\ 011)_2$

Addition of the sign bits to form the LSH and MSH yields:

LSH of (K) x (Y) = $(100\ 110\ 011\ 001)_2 = (4633)_8\ (-4)_{10}$

MSH of (K) x (Y) = $(111\ 111\ 111\ 111)_2 = (7777)_8\ (-0)_{10}$

Thus, if the constant K is entered as an integer, with the radix point (implied) located to the right of the least significant octal digit, then the desired 12-bit product is located in the LSH of the two word product obtained from the arithmetic unit and no further operations are required to retrieve the 12-bit product.

B. Example 2. Consider the product (K) x (Y), where $K = (0.25)_{10}$ and $Y = (+4)_{10} = (3144)_8$ after analog-to-digital conversion. Using method (2) above to enter K into the 160 computer:

$$K = (0.25)_{10} = (2000)_8.$$

The 168 arithmetic unit will form the product $(2000)_8 \times (3144)_8$ as follows:

$(2000)_8 - \text{sign bit} = (01\ 000\ 000\ 000)_2$ (sign bit = 0)

$(3144)_8 - \text{sign bit} = (11\ 001\ 100\ 100)_2$ (sign bit = 0)

$\text{22-bit product} = (0\ 110\ 011\ 001\ 000\ 000\ 000\ 000)_2$

Addition of the sign bits to form the LSH and the MSH yields:

$$\text{LSH of } (K) \times (Y) = (000\ 000\ 000\ 000)_2 = (0000)_8 = (0)_{10}$$

$$\text{MSH of } (K) \times (Y) = (001\ 100\ 110\ 010)_2 = (1462)_8 = (+2)_{10}$$

If the MSH is now right-shifted one binary bit, i. e.,

$$\text{MSH' of } (K) \times (Y) = (000\ 110\ 011\ 001)_2 = (+1)_{10},$$

then we have the desired 12-bit answer in the MSH of the two word product received from the arithmetic unit. Further difficulties arise using this method, however, due to the use of bit 11 as the sign bit in the 160 computer word. For example, if $K = (+0.5)_{10}$, then the octal representation of K is $(4000)_8$. Both the computer and the arithmetic unit interpret this number as a negative number since bit 11 is a "1". Hence the usable positive fractions are restricted to be less than one-half.

C. Example 3. Consider the product (K) x (Y), where $K = (2.5)_{10}$ and $Y = (2)_{10} = (1462)_8$. Using method (3) above to enter K into the 160 computer,

$$K = (2.5)_{10} = (2400)_8.$$

The arithmetic unit will form the product $(2400)_8 \times (1462)_8$ as follows:

$$(2400)_8 - \text{sign bit} = (10\ 100\ 000\ 000)_2 \qquad (\text{sign bit} \div 0)$$

$$(1462)_8 - \text{sign bit} = (01\ 100\ 110\ 010)_2 \qquad (\text{sign bit} = 0)$$

$$\text{22-bit product} = (0\ 011\ 111\ 111\ 101\ 000\ 000\ 000)_2$$

Addition of the sign bits forming the LSH and the MSH:

$$\text{LSH of } (K) \times (Y) = (001\ 000\ 000\ 000)_2 = (1000)_8$$

$$\text{MSH of } (K) \times (Y) = (000\ 111\ 111\ 111)_2 = (0777)_8$$

At the completion of the multiply operation, the LSH is located in storage cell n and the MSH is located in cell n+1 of the 160 computer, i. e.,

(MSH)				(LSH)			
0	7	7	7	1	0	0	0
000	111	111	111	001	000	000	000
Cell n+1				Cell n			

The desired answer $(3775)_8$ can be "uncovered" by the removal of bit 11 of the LSH (the sign bit added by the arithmetic unit) and a "long right inter-cell" shift of one binary bit in the two cells:

000	111	111	111	001	000	000	000
000	011	111	111	101	000	000	000
0	3	7	7	5	0	0	0
(MSH)				(LSH)			

However, the answer is split between two storage cells. A series of shifting and masking operations must be performed on these two words to place the desired answer in one 12-bit storage cell. This problem could be greatly simplified by the incorporation of a long right shift capability in the 168 arithmetic unit, however, at present, this shifting must be done in the 160 computer.

D. Example 4. Consider the product (K) x (Y), where both K and Y are expressed in the number system shown in Table 2-1, i.e., in the analog-digital conversion number system. Let $K = Y = (+2)_{10} = (1462)_8$. Then the arithmetic unit will form the product (K) x (Y) as follows:

$$\text{LSH of } (K) \times (Y) = (010\ 111\ 000\ 100)_2 = (2704)_8$$

$$\text{MSH of } (K) \times (Y) = (000\ 101\ 000\ 110)_2 = (0506)_8$$

The desired answer $(3144)_8$ does not appear in either the LSH or the MSH. However, if this two word product is divided by $(0631)_8$ (which corresponds to unity in Table 2-1), then the desired product will appear as the quotient. Referring to Figure I-2, let the dividend by $(05062704)_8$ and the divisor be $(0631)_8$. Then the arithmetic unit performs the short division in the following steps:

$$\text{dividend}\begin{cases}(2704)_8 \text{ - sign bit} = (10\ 111\ 000\ 100)_2 & \text{(sign bit = 0)}\\ (0506)_8 \text{ - sign bit} = (00\ 101\ 000\ 110)_2 & \text{(sign bit = 0)}\end{cases}$$

$$\text{divisor} — (0631)_8 \text{ - sign bit} = (00\ 110\ 011\ 001)_2 \qquad \text{(sign bit = 0)}$$

$$\text{11-bit quotient} = (11\ 001\ 100\ 100)_2$$

$$\text{11-bit remainder} = (00\ 000\ 000\ 000)_2$$

Addition of sign bits to quotient and remainder yields:

$$\text{quotient} = (011\ 001\ 100\ 100)_2 = (3144)_8 = (+4)_{10}$$

$$\text{remainder} = (000\ 000\ 000\ 000)_2 = (0000)_8 = (0)_{10}$$

By means of as short multiply operation followed by a short divide operation, the product $(2)_{10} \times (2)_{10}$ has been accomplished with the desired answer in a single 12-bit storage cell in the 160 computer. Note that this computation can be expressed as:

$$\frac{(1462) \times (1462)}{(0631)} = \frac{(1462) \times [(2) \times (0631)]}{(0631)}$$

Thus if all arithmetic operations involving quantities which have undergone A/D conversion are done in the "analog-to-digital psuedo-number system" as shown in Table 2-1, then the CDC 160 programming procedures are greatly simplified. Any constants entered into the computer by the user must either be entered into the computer in the psuedo-number system format (Table 2-1), or must be converted to this format by appropriate programming. This latter method was chosen for work done in this paper.

APPENDIX II

EXTERNAL FUNCTION CODES FOR THE CDC 160 COMPUTER

II-1. EXTERNAL FUNCTION CODES.

The CDC 160 computer machine language code for selection of external equipment is:

CELL	CONTENTS	OSAP MNEMONIC	
n	7500	EXF	
n+1	xxxx		xxxx

where the 12-bit operand, xxxx, is known as an external function code. The external function command as shown above is used to select an external device to perform some specific function depending upon the external function code used. A partial list of external function codes is given in Table II-1.

Only one external device may be selected at any one time. Selection of any device automatically disconnects any other selected device. If an illegal selection is attempted the computer will be indefinitely delayed. Appropriate input-output instructions, if required, must follow the external function command.

External Device	Function Desired	External Function Code	Remarks
168 arithmetic unit	Short divide	3300	
168 arithmetic unit	Short multiply	3301	
168 arithmetic unit	Long divide	3302	
168 arithmetic unit	Long multiply	3303	
ADC-DAC	A/D conversion	140y	See note 1.
ADC-DAC	D/A conversion	24xy	See notes 1,2.

Table II-1. CDC 160 External Function Codes.

NOTE 1. For the ADC-DAC external function codes, y is the channel number.

NOTE 2. The octal number x controls a relay in the ADC-DAC which is available for control of external equipment. As used herein, this relay is connected in series with the "OPERATE-RESET" switch of the Pace TR-20

analog computer (as shown in Figure II-1) to permit the CDC 160 digital computer to control the analog computer. If the digit, x, in the external function code 24xy is "1", then the relay switches the analog computer to the OPERATE mode; if x is "0", the analog computer is switched to the RESET mode. The MASTER CLEAR key on the 160 computer console will also reset the analog computer.

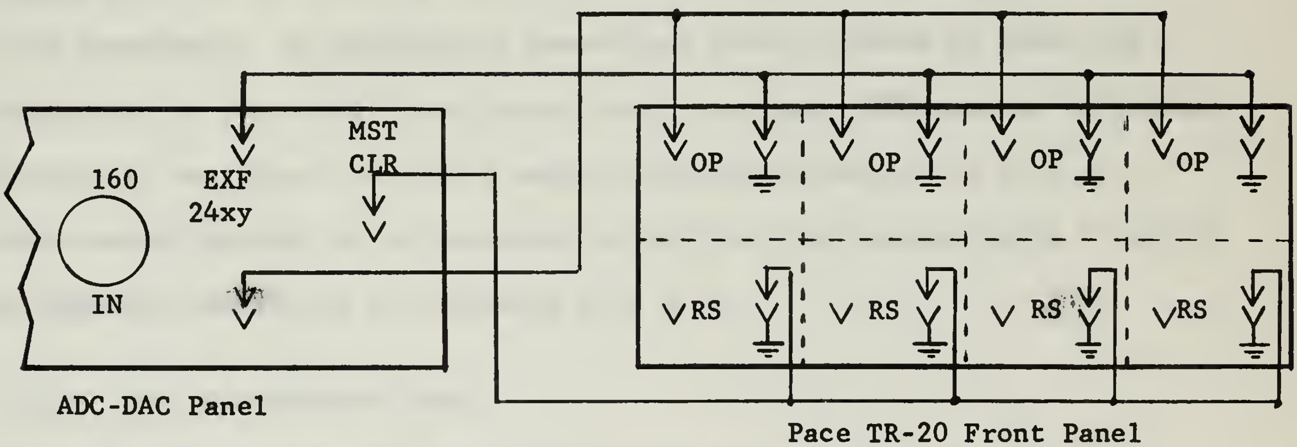


Figure II-1. Connections for Remote Control of Analog Computer

APPENDIX III

SUBROUTINE KMOD

III-1. GENERAL DESCRIPTION.

Subroutine KMOD is designed to convert a series of octal numbers to a format compatible with the CDC 160 computer, the CDC 168 arithmetic unit and the D/A-A/D conversion equipment. Conversion is accomplished by multiplication of the number by $(0631)_8$ using the CDC 168 arithmetic unit. After conversion, the numbers are of the same format as the output quantities of the ADC. Further arithmetic operations involving these numbers and A/D output quantities can then be accomplished with greatly simplified programming procedures. An abbreviated conversion table is shown in Table III-1. Comparison of Table III-1 and Table 1 will show that KMOD acts as a "analog-to-digital converter" for these numbers (designated herein as K'). As with analog voltages to be converted in the ADC, the numbers to be converted by KMOD are restricted to the range $|K'| \leq 5$.

III-2. USE OF SUBROUTINE KMOD.

A. Calling sequence. This subroutine may be called from the main program with the following two lines of coding:

Machine Code	Mnemonic	Remarks
0101	SHA 01 (or PTA)	Shift contents of P-register to the A-register
7000	JPI 00	Indirect jump to address located in cell 0000.

The subroutine generates and stores its own linkage address to the main program.

B. Storage Requirements. KMOD requires 161 (octal) storage cells. KMOD may be loaded at any location in the 160 memory.

C. Data Entry. The following data must be entered into the CDC 160 computer manually in the locations shown:

Cell Number	Contents
0000	Load address of subroutine KMOD.
0001	Number of constants (K') to be converted. (24 maximum).
0002	Storage address of first converted constant, K. See note 1.
0003	Sign of $K'_1 = \begin{cases} 0000 \text{ if } K'_1 \text{ is positive} \\ 0001 \text{ if } K'_1 \text{ is negative.} \end{cases}$
0004	K'_1. See note 2.
0005	Sign of K'_2.
0006	K'_2.
0007	Sign of K'_3.

.
.
.
etc.

Note 1. If more than one K' is to be converted, then the converted numbers (K_i) are stored in every other storage cell beginning with the storage location entered in cell 0002. This permits greater ease in input-output operations in the main program.

Note 2. The constants to be converted must be entered into the 160 computer in the format $(x.xxx)_8$. Example: If $K' = (1.4325)_8$, this would be entered into the 160 computer as $(1433)_8$.

Manually entered data is not destroyed during execution of KMOD. Thus, execution can be repeated without re-entry of data.

D. Low core cell usage. In addition to the low core cells used for manual entry of data, cells 0064 thru 0077 are used during the execution of this subroutine. After execution of KMOD, cells 64 thru 77 are available for other purposes.

Octal Number to be converted (K')	Octal Number after conversion (K)
-5.000	4002
-4.400	4317
-4.000	4633
-3.400	5150
-3.000	5464
-2.400	6001
-2.000	6315
-1.400	6632
-1.000	7146
-0.400	7463
0.000	7777 or 0000
0.400	0314
1.000	0631
1.400	1145
2.000	1462
2.400	1776
3.000	2313
3.400	2627
4.000	3144
4.400	3460
5.000	3775

Table III-1. Conversion Table for Subroutine KMOD.

III-3. DETAILED DESCRIPTION OF KMOD.

If the arguments of the short multiply operation are of the format given in the preceeding instructions, then, as shown in Example 3, Appendix I, the desired 12-bit product is split between two successive storage locations in the 160 computer. Let the two-word product as received from the arithmetic unit be expressed as follows:

zxx	xxx	xxx	xxy	wxx	xxx	xxx	xxx	(binary)
m'	d'_1	d'_2	d'_3	d'_4	n	n	n	(octal)
MSH				LSH				

where x = either binary digit (1 or 0)

y = 11th bit of the 22-bit product

w and z = sign bits inserted into the 22-bit product by the arithmetic unit.

The following procedures are necessary to retrieve the desired 12-bit product, $K = (k_1k_2k_3k_4)_8$.

A. Removal of the sign bit in the LSH. Since w has been inserted in the middle of the desired product, it must be removed. To do this:

1. Set w equal to y.
2. Shift MSH one bit to the right (this requires an eleven bit left shift in the computer).

Upon completion of step 2, the two words are:

yzx	xxx	xxx	xxx	yxx	xxx	xxx	xxx
m	k_1	k_2	k_3	k_4	n	n	n
MSH				LSH			

B. Round-off correction. The radix point of the two-word product is implied after octal digit K_4. If the digits $(nnn)_8$ are greater

than $(444)_8$, then k_4 must be increased by one unit. However, if $k_4 = (7)_8$ then this round-off correction will propagate to k_3, requiring that k_3 be increased by one unit. If $(nnn)_8$ is less than $(444)_8$, no round-off correction is necessary. Thus the steps in the round-off correction are:

1. Determine if $(nnn)_8$ is greater than $(444)_8$. If not, skip the following steps.
2. If round-off correction is necessary, determine if k_4 is equal to $(7)_8$. If so, add one to k_3. If not, go to next step.
3. Add one to k_4.

C. Removal of extraneous bits. Digits m and n have now served their purpose and can be discarded. To do this:

1. Mask off m and shift MSH left three bits.
2. Mask off (nnn) and shift LSH left three bits.

   Upon completion of this step, the two words are:

k_1	k_2	k_3	0	0	0	0	k_4
MSH				LSH			

3. Add LSH to MSH to get desired number $K = (k_1\ k_2\ k_3\ k_4)_8$.

III-4. PROGRAM LISTING FOR SUBROUTINE KMOD.

The following pages give the OSAP program listing for subroutine KMOD. Sufficient comments are inserted to make the program self-explanatory.

SUBROUTINE KMOD

Cell	Mach Code		Symbolic		Remarks
0	0602		adn	02	Form link address to main program.
1	4072		std	link	Store link address.
2	2000		ldd	00	Load the address of KMOD.
3	0620		adn	20	Form recycle address for multiple K's.
4	4071		std	repadd	Store recycle address.
5	0403		ldn	03	Load initial sign address.
6	4064		std	sinadd	Store initial sign address.
7	0404		ldn	04	Load initial K' address.
10	4066		std	kadd	Store initial K' address.
11	2001		ldd	01	Load number of (K')'s to be converted.
12	4070		std	index	Store to form index.
13	2002		ldd	02	Load storage address for 1st converted K'.
14	4067		std	stoadd	Store above address.
15	2200		ldf		Load 0631.
16	0631			631	
17	4074		std	one	Store as argument for short multiply.

The preceding instructions perform several book-keeping tasks prior to the execution of the subroutine. All of the manually entered data is relocated so that upon execution of the following instructions, this manually entered data is not destroyed. The cell numbering is given relative to the initial load address.

Cell	Mach Code		Symbolic		Remarks
20	2164		ldi	sinadd	Load sign of K' to be converted.
21	4073		std	sign	Store.
22	2166		ldi	kadd	Load K' to be converted.
23	6206		pjf	h	If bit 11 is 0, jump.
24	2200		ldf		If bit 11 is 1, load the number 2000.
25	2000			2000	
26	4065		std	temp	Temporarily store 2000.
27	2166		ldi	kadd	Reload K'.
30	3465		sbd	temp	Subtract 2000 from K'. See *.
31	4075	h	std	outbuf	Store in output location.
32	7500		exf		Select a.u., short multiply.
33	3301			3301	
34	7306		out	ki	Output 0631 and K'.
35	0076			76	
36	7205		inp	kia	Input two word product, LSH in cell 76,
37	0100			100	MSH in cell 77.
40	6105		nzf	g	Unconditional jump around table.
41	6004		zjf	g	
42	0074	ki		74	
43	0076	kia		76	Input-output table.
44	4000	con		4000	

The sign of the K' under conversion has been determined and stored. The two word product has been received from the arithmetic unit.

Cell	Mach Code	Symbolic			Remarks
					* Note: If K' > 3777, then bit 11 = 1, but the sign function has been taken away from bit 11 since the sign of K' is entered as a separate word. Since the maximum value of K' is 5000, then subtraction of 2000 will yield a positive number. A later operation adds back this quantity.
45	2077	g	ldd	msh	Load MSH of two-word product.
46	0201		lpn	01	Mask off bits 01 thru 11.
47	6111		nzf	c	If bit 00 is 1, jump.
50	2304		ldb	con	If bit 00 is 0, load number 4000.
51	1076		lpd	lsh	Take logical product of LSH and 4000.
52	6014		zjf	d	If bit 11 of LSH is 0, no action required.
53	2707		lcb	con	If bit 11 of LSH is 1, load 3777.
54	1076		lpd	lsh	Make bit 11 of LSH equal to 0.
55	4076		std	lsh	Store modified LSH.
56	6310		njf	d	Unconditional jump.
57	6007		zjf	d	
60	2114	c	ldb	con	If bit 00 of MSH is 1, load 4000.
61	1076		lpd	lsh	Is bit 11 of LSH 1 or 0?
62	6106		nzf	d	If 1, no action required, jump.
63	2317		ldb	con	If 0, load 4000.
64	3076		add	lsh	Make bit 11 of LSH 1 by adding 4000.
65	4076		std	lsh	Store modified LSH.
					This last sequence of instructions has made the most significant bit (bit 11) of the LSH agree with the least significant bit (bit 00) of the MSH, thus removing the sign bit inserted in the two word product by the arithmetic unit.
66	2077	d	ldd	msh	Load MSH.
67	0111		sha	11	Left shift MSH 11 bits. This moves bit
70	0110		sha	10	00 to the 11th bit position for later
71	0103		sha	03	removal (since it is no longer needed).
72	4077		std	msh	Store modified MSH.
73	2200		ldf		Load the number 0777.
74	0777			777	
75	1076		lpd	lsh	Mask off upper three bits of LSH.
76	4065		std	temp	Store lower nine bits.
77	2200		ldf		Load the number 0400.
100	0400			400	
101	3465		sbd	temp	Subtract lower nine bits of LSH.
102	6216		pjf	f	If result positive, no round off required.
103	2200		ldf		If negative, load number 7000.
104	7000			7000	
105	1076		lpd	lsh	Mask off lower nine bits of LSH.
106	4076		std	lsh	Store modified LSH.
107	2200		ldf		Reload number 7000.
110	7000			7000	

Cell	Mach Code	Symbolic			Remarks
111	3476		sbd	lsh	Subtract LSH, to see if round-off correction
112	6102		nzf	e	will propagate to MSH. If not, jump.
113	5477		aod	msh	If so, add 1 to MSH.
114	2200	•	ldf		Load number 1000.
115	1000			1000	
116	3076		add	lsh	Add 1000 to LSH for round-off correction.
117	4076		std	lsh	Store modified LSH.

Since the least significant 9 bits of the LSH are discarded in the condensation of the two word product to a 12-bit word, a test is made to determine if these bits will force a round-off correction to be made. If a round-off correction is to be made, a further test is required to determine if this correction will propagate to the MSH. Corrections are then made to the LSH and MSH if necessary.

Cell	Mach Code	Symbolic			Remarks
120	2200	f	ldf		Load the constant 0777.
121	0777			777	
122	1077		lpd	msh	Mask off upper 3 bits of MSH.
123	0110		sha	10	Shift MSH left three bits.
124	4077		std	msh	Store modified MSH.
125	2076		ldd	lsh	Load LSH.
126	0110		sha	10	Shift LSH left three bits.
127	4076		std	lsh	Store.
130	0407		ldn	07	Load the number 0007.
131	1076		lpd	lsh	Mask off upper nine bits of LSH.
132	3077		add	msh	Add MSH to LSH to form K.
133	4167		sti	stoadd	Store K.
134	2166		ldi	kadd	Reload K' (unconverted).
135	6205		pjf	i	If positive, jump.
136	2200		ldf		If negative, load the number 1462.
137	1462			1462	
140	3167		adi	stoadd	Add "2" to converted K.
141	4167		sti	stoadd	Store.
142	2073	i	ldd	sign	Load sign of K'.
143	6003		zjf	ind	If positive, no action required.
144	2567		lci	stoadd	If negative, complement K.
145	4167		sti	stoadd	Store K.

The extraneous bits in the LSH and the MSH are set to zero and the remaining bits are shifted so that the LSH and the MSH can be added to form the 12-bit converted K. If 2000 was subtracted earlier, it is added back to the K in the form 1462. The sign of K is made to agree with that of K'.

[illegible]

Cell	Mach Code	Symbolic			Remarks
146	2070	ind	ldd	index	Load number of (K')'s to be converted.
147	0701		sbn	01	Subtract 1.
150	4070		std	index	Store.
151	6010		zjf	j	If zero, return to main program.
152	0402		ldn	02	If not zero, load 0002.
153	5067		rad	stoadd	Advance storage address by 2.
154	0402		ldn	02	
155	5064		rad	sinadd	Advance sign address.
156	0402		ldn	02	
157	5066		rad	kadd	Advance address of K' to be converted.
160	7071		jpi	repadd	Repeat subroutine for next K'.
161	7072	j	jpi	link	Return to main program.
	0064	sinadd	equ	64	Address of the sign of K'.
	0065	temp	equ	65	Temporary storage cell.
	0066	kadd	equ	66	Address of K'.
	0067	stoadd	equ	67	Storage address of K.
	0070	index	equ	70	Number of (K')'s to be converted.
	0071	repadd	equ	71	Address for subroutine repeat.
	0072	link	equ	72	Link address to main program.
	0073	sign	equ	73	Sign of K' presently under conversion.
	0074	one	equ	74	0631.
	0075	outbuf	equ	75	Output location for K'.
	0076	lsh	equ	76	LSH of two word product.
	0077	msh	equ	77	MSH of two word product.
	0000		end		

APPENDIX IV

PROGRAM DIGIT

IV-1. GENERAL DESCRIPTION.

Program DIGIT is a CDC 160 digital computer program written for the control of the simulated hybrid system discussed in Chapter IV. A block diagram of the simulated system is shown in Figure 4-6. Specifically, DIGIT computes

$$U(k) = +a_1 [X_1(k) - r_o - \dot{r}t(k)] + a_2[X_2(k) - \dot{r}] \qquad \text{(IV-1)}$$

where $X_1(k)$, $X_2(k)$, $\dot{r}t(k)$ and $\dot{r}$ are digital representations (in the analog digital psuedo-number system) of sampled analog quantities received from the ADC and a_1, a_2 and r_o are manually-entered quantities. The quantity $U(k)$ is transmitted to the DAC after computation is completed. All of the quantities in Equation IV-1 are further defined in Chapter IV.

DIGIT uses subroutine KMOD to convert the quantities a_1, a_2 and r_o from the normal octal representation of a decimal number to a form consistent with the analog-digital psuedo-number system generated by the A/D-D/A conversion process. All arithmetic operations are then carried out in this number system using the CDC 168 arithmetic unit for multiplication and division.

For brevity, the following terminology is used in this discussion:

(1) $SUM1 = X_1(k) - r_o - \dot{r}t(k)$

(2) $SUM2 = X_2(k) - \dot{r}$.

(3) $P_1 = PROD1 = +a_1(SUM1)$.

(4) $P_2 = PROD2 = +a_2(SUM2)$.

(5) $SUM3 = P_1 + P_2 = U(k)$.

The following features are included in DIGIT:

(1) Overflow tests are performed on P_1, P_2 and SUM3 and if overflow occurs, automatic overflow correction or problem termination is effected at the option of the user. Problem termination includes the "resetting" of the analog computer and a visual indication of the quantity which overflowed.

(2) A variable length time-delay loop is included permitting internal control of the sampling rate. This loop may be bypassed entirely if external timing control is desired.

(3) The operate-reset modes of the PACE TR-20 analog computer may be controlled remotely by DIGIT.

The various control options and data for subroutine KMOD are entered manually in low core cells. Execution of the program does not destroy this manually entered data, hence the program may be re-run without re-entry of data.

DIGIT is written in a general format so that it may be used with any analog computer simulation of a second-order system where the control law consists of a linear combination of the states of the system. Any of the quantities in Equation IV-1 may be set to zero either by setting the particular A/D channel at minus five volts for the sampled quantities or entering a zero for the manually entered quantities.

IV-2 USE OF PROGRAM DIGIT.

A. LOAD program DIGIT and subroutine KMOD where desired in the CDC 160 computer memory. DIGIT requires 327 (octal) storage cells; KMOD requires 161 (octal) cells. In addition, low core cells 0000 thru 0013 and 0040 thru 0077 are used by DIGIT.

B. ENTER the following data MANUALLY into the indicated locations:

CELL NUMBER	CONTENTS
0000	Load address of KMOD.
0001	0003 (number of conversions for KMOD).
0002	0040 (memory address of r_o after conversion by KMOD).
0003	sign of $r_o = \begin{cases} 0000 \text{ if } r_o \text{ is positive.} \\ 0001 \text{ if } r_o \text{ is negative.} \end{cases}$
0004	$\lvert r_o \rvert$ in x.xxx format ($\lvert r_o \rvert \leq 5$).
0005	Sign of $a_1 = \begin{cases} 0000 \text{ if } a_1 \text{ is positive.} \\ 0001 \text{ if } a_1 \text{ is negative.} \end{cases}$
0006	$\lvert a_1 \rvert$ in x.xxx format ($\lvert a_1 \rvert \leq 5$).
0007	Sign of $a_2 = \begin{cases} 0000 \text{ if } a_2 \text{ is positive.} \\ 0001 \text{ if } a_2 \text{ is negative.} \end{cases}$
0010	$\lvert a_2 \rvert$ in x.xxx format ($\lvert a_2 \rvert \leq 5$).
0011	$\begin{cases} 0000 \text{ if time delay loop is to be bypassed.} \\ 0001 \text{ if time loop is to be used.} \end{cases}$
0012	Coarse delay factor. See discussion below.
0013	Fine delay factor. See discussion below.

C. External Equipment. DIGIT requires CDC 160 access to the following external equipment:

(1) CDC 168 arithmetic unit.

(2) A/D-D/A conversion equipment with

(a) X_1 available on channel #1, A/D.

(b) X_2 available on channel #2, A/D.

(c) $\dot{r}t(k)$ available on channel #3, A/D.

(d) $\dot{r}$ available on channel #4, A/D.

(e) channel #1, D/A available for CDC 160 output.

(f) all input/output quantities appropriately biased so as to lie within zero to minus ten volts. A common bias is recommended.

(3) Remote control connections of the operate-reset switch of the PACE TR-20 analog computer (optional).

D. RUN DIGIT from the initial load address.

IV-3 OVERFLOW PROVISIONS.

Product overflow detection is done by comparison of the product magnitude $|a_i \times SUM_i|$ and the variable magnitude $|SUM_i|$, provided that a_i is greater than unity. If the product magnitude is less than the variable magnitude, overflow has occurred. No test is made if a_i is less than unity since overflow is not then possible. If overflow is detected, the computer will either error halt (providing a visual indication as to which product overflowed) or will substitute the maximum possible value (4000 or 3777) for the intended product depending upon the signs of a_i and SUM_i. If the error halt option is used, the analog computer is reset to the initial conditions.

Summation overflow detection is accomplished by detection of the sign change of SUM3 as overflow occurs, provided that both arguments of the sum agree in sign. If the arguments are of opposite sign, no overflow is possible. The same provisions concerning error halt or overflow correction are provided as for product overflow.

If the error halt option is selected, an indication of the quantity causing the overflow is displayed in the A-register of the 160 computer. The error halt may be bypassed by manual entry of the appropriate error bypass code into the A-register (after clearing of the registers) and re-running of the program. This will provide automatic overflow correction. The error halt indications and error bypass codes are tabulated in Table IV-1.

A-Register Display	Quantity Causing Overflow	Error Halt Number
0001	a_1P_1	1
0002	a_2P_2	2
0003	SUM3	3

To Bypass Error Stop #	Enter in A-Register
1	0001
2	0002
3	0004
1 & 2	0003
1 & 3	0005
2 & 3	0006
all	0007

Table IV-1. Overflow Error Halt and Bypass Table.

IV-4. INTERNAL TIMING CONTROL.

The variable-length time delay loop in program DIGIT is shown in flow chart form in Figure IV-1. Delay is effected by loading the fine delay factor into the arithmetic register of the CDC 160 computer and counting this number down to zero by successive subtraction of the number one. This count-down procedure is repeated as many times as is indicated by the coarse delay factor. A total of 52.4 milliseconds is required to count down the number 7777 to zero. A suggested method for accomplishing a time delay of T milliseconds is:

(a) Set the fine delay factor to 7777.

(b) Set the coarse delay factor to the first octal digit exceeding T/52.4. (T in milliseconds). This should yield a time delay slightly over T milliseconds in length. Using an external time-measuring device (oscilloscope or Brush recorder), adjust the fine delay factor downward to yield the desired time delay.

Example: To achieve a one second sample interval for program DIGIT, the coarse delay factor was set to 23_8 and the fine delay factor was adjusted to 7600_8.

IV-5 EXTERNAL TIMING CONTROL.

Insertion of zero in cell 0011 will bypass the internal timing loop of program DIGIT. External timing control may be accomplished by the connection of an external clock to the "input disable" jack of the ADC. This jack is connected via an AND-gate to the "input ready" line of the 160 computer input cable. The 160 computer will delay execution of the input instruction until the "input ready" line drops to -13 volts. Thus, if the "input disable" is held at ground level by the external clock but periodically dropped to -13 volts at the sampling frequency, sample delay is accomplished. Since input control is accomplished on the "input ready" line vice the "input request" line, the first sample that is inputted to the 160 must be discarded since this sample is stale. Also, the "input disable" must be held at -13 volts long enough for four channels to be sampled each period (about 450 microseconds).

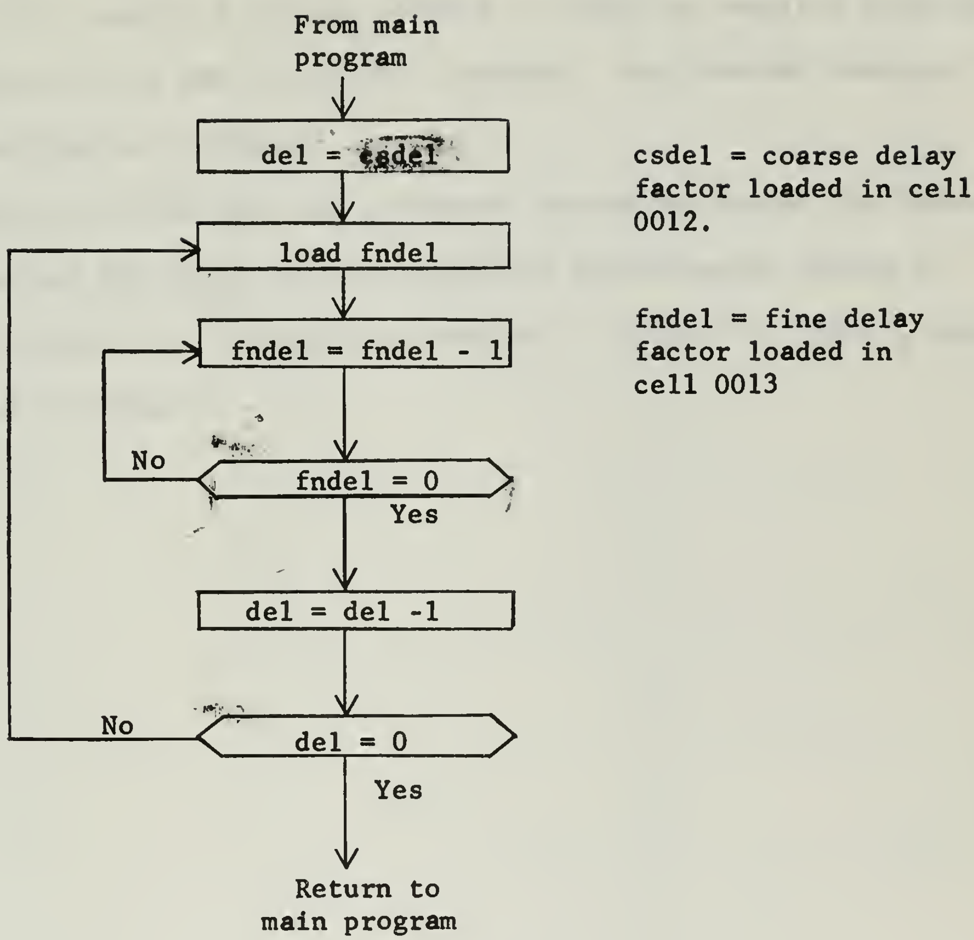
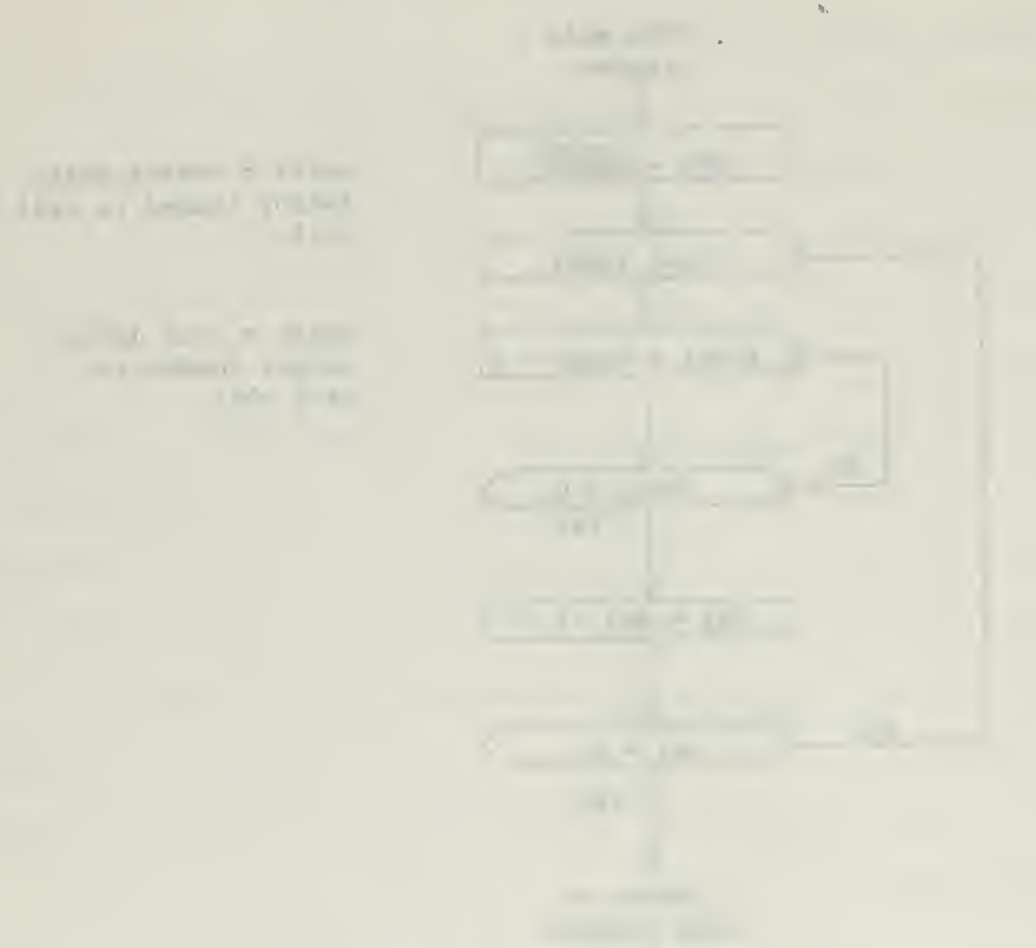


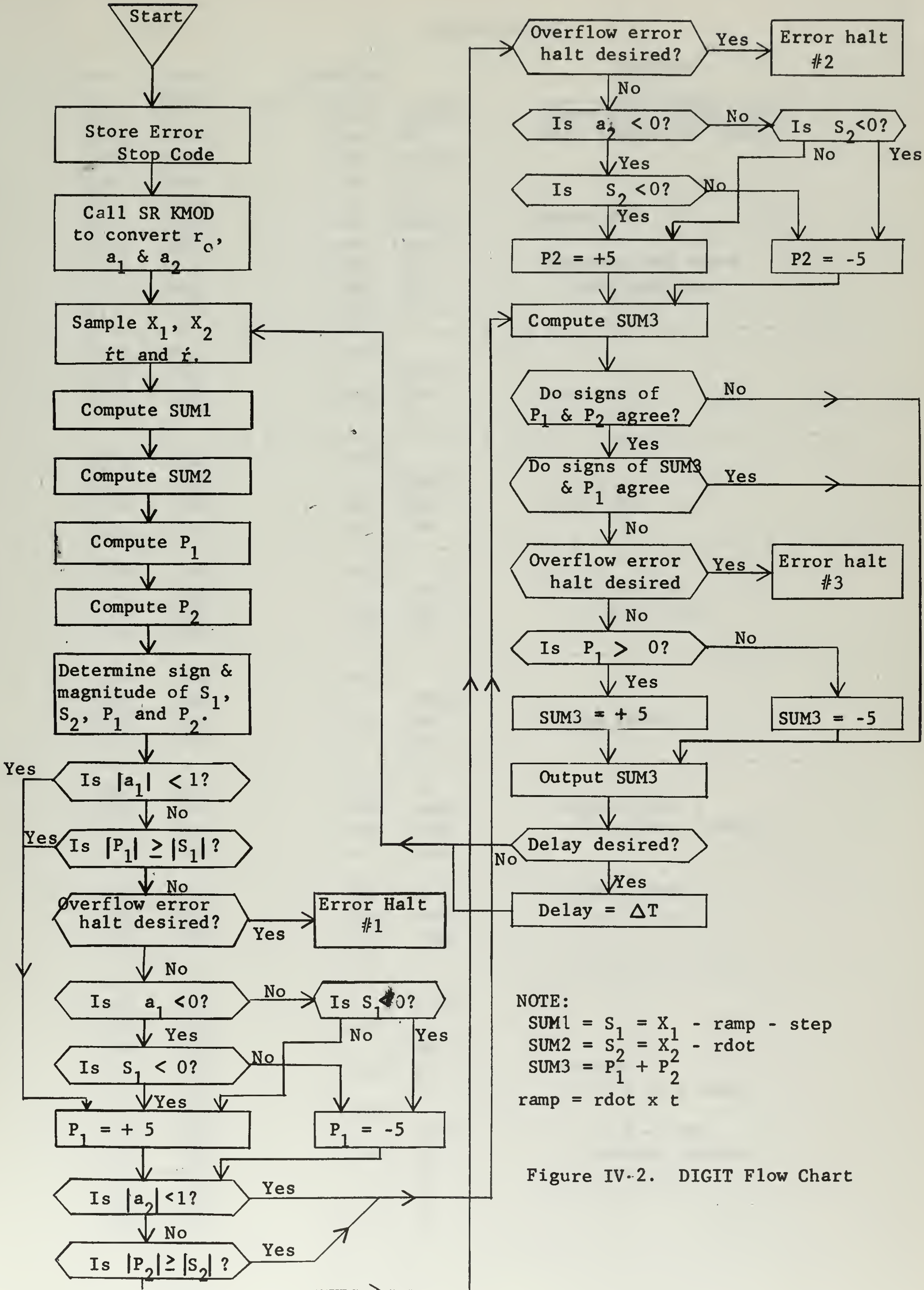
Figure IV-1. Program DIGIT Variable-Length Time Delay Loop



IV-6. DIGIT Flow Chart and Program Listing.

Figure IV-2 is a flow chart presentation of Program DIGIT. Pages 72 thru 77 contain a program listing of DIGIT as compiled using the OSAP compiler for the CDC 160 digital computer. Both machine language and mnemonic listings are presented.

It should be noted that all arithmetic operations within the sample loop are carried out using the analog-digital psuedo-number system as explained in Chapter 3. This method requires a division by $(0631)_8$ after each product is formed.



NOTE:

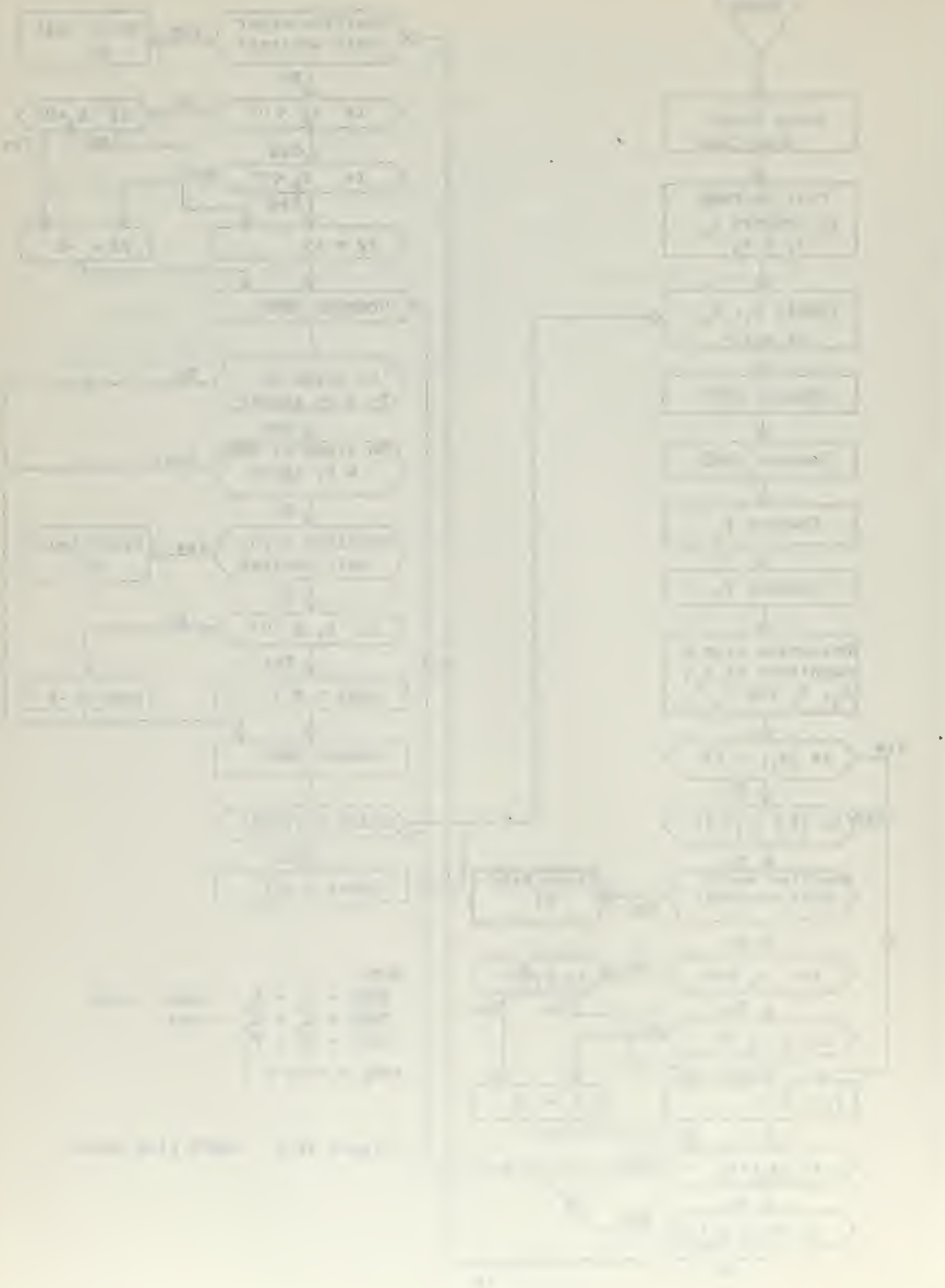
SUM1 = $S_1 = X_1$ - ramp - step

SUM2 = $S_2 = X_2$ - rdot

SUM3 = $P_1 + P_2$

ramp = rdot x t

Figure IV-2. DIGIT Flow Chart



PROGRAM DIGIT

```
Cell   Mach     Symbolic              Remarks
       Code
0100   4047     std   errflg          Store error stop code.
0101   0101     sha   01              Call subroutine KMOD.
0102   7000     jpi   sradd
0103   2200     ldf
0104   0631           631
0105   4062     std   onea            Store 0631.
0106   4065     std   oneb
0107   0101     sha   01              Generate and store
0110   0603     adn   03                 loop address.
0111   4075     std   loop
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0112   7500     exf
0113   1401           1401
0114   7265     inp   ch1             Input X1.
0115   0077           77
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0116   7500     exf
0117   1402           1402
0120   7262     inp   ch2             Input X2.
0121   0100           100
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0122   7500     exf
0123   1403           1403            Input ramp =
0124   7257     inp   ch3                rdot x t(k).
0125   0075           75
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0126   7500     exf
0127   1404           1404
0130   7254     inp   ch4             Input rdot.
0131   0074           74
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0132   2076     ldd   x1
0133   3474     sbd   ramp            Compute sum1 =
0134   3440     sbd   step               X1 - ramp - step.
0135   4043     std   sum1
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0136   2077     ldd   x2              Compute sum2 =
0137   3473     sbd   rdot               X2 - rdot.
0140   4045     std   sum2
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0141   7500     exf
0142   3301           3301
0143   7330     out   a1
0144   0044           44
0145   7230     inp   p1              Compute
0146   0062           62                 P1 = a1 x sum1.
0147   7500     exf
0150   3300           3300               a1 = -1 for
0151   7324     out   p1                 optimum control.
0152   0063           63
0153   7224     inp   q1
0154   0070           70
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

[illegible]

PROGRAM DIGIT (con't)

Cell	Mach Code	Symbolic			Remarks
0155	7500		exf		
0156	3301			3301	
0157	7315		out	a2	
0160	0046			46	
0161	7215		inp	p2	Compute
0162	0065			65	$P_2 = a_2 \times$ sum2.
0163	7500		exf		
0164	3300			3300	$a_2 = -1.5$ for
0165	7311		out	p2	optimum control.
0166	0066			66	
0167	7211		inp	q2	
0170	0072			72	
0171	6114		nzf	a	
0172	6013		zjf	a	
0173	0042	a1		42	
0174	0044	a2		44	
0175	0060	p1		60	
0176	0063	p2		63	
0177	0066	q1		66	
0200	0070	q2		70	Input-output table.
0201	0076	ch1		76	
0202	0077	ch2		77	
0203	0074	ch3		74	
0204	0073	ch4		73	
0205	2043	a	ldd	sum1	
0206	6205		pjf	b	Determine sign and
0207	2443		lcd	sum1	magnitude of sum1.
0210	4057		std	mags1	
0211	0401		ldn	01	
0212	6103		nzf	t	
0213	4057	b	std	mags1	
0214	0400		ldn	00	
0215	4056	t	std	sins1	
0216	2045		ldd	sum2	
0217	6205		pjf	c	
0220	2445		lcd	sum2	Determine sign and
0221	4055		std	mags2	magnitude of sum2.
0222	0401		ldn	01	
0223	6103		nzf	u	
0224	4055	c	std	mags2	
0225	0400		ldn	00	
0226	4054	u	std	sins2	
0227	2066		ldd	prod1	
0230	6205		pjf	d	
0231	2466		lcd	prod1	Determine sign and
0232	4053		std	magp1	magnitude of P_1.
0233	0401		ldn	01	
0234	6103		nzf	v	
0235	4053	d	std	magp1	
0236	0400		ldn	00	
0237	4052	v	std	sinp1	

[illegible]

Cell	Mach Code		Symbolic		Remarks
0240	2070		ldd	prod2	
0241	6205		pjf	e	
0242	2470		lcd	prod2	Determine sign and
0243	4051		std	magp2	magnitude of P_2.
0244	0401		ldn	01	
0245	6103		nzf	w	
0246	4051	e	std	magp2	
0247	0400		ldn	00	
0250	4050	w	std	sinp2	
0251	2006		ldd	alu	
0252	1252		lpf	const	
0253	6033		zjf	f	
0254	2053		ldd	magp1	P_1 overflow test.
0255	3457		sbd	mags1	
0256	6230		pjf	f	
0257	6027		zjf	f	
0260	2047		ldd	errflg	
0261	0201		lpn	01	
0262	6105		nzf	g	
0263	7500		exf		
0264	2401			2401	Error stop 1 if
0265	0401		ldn	01	P_1 overflows.
0266	0000		err		
0267	2056	g	ldd	sins1	
0270	6004		zjf	h	
0271	2005		ldd	sina1	
0272	6004		zjf	cc	
0273	6107		nzf	dd	
0274	2005	h	ldd	sina1	
0275	6005		zjf	dd	
0276	0401	cc	ldn	01	P_1 overflow correction.
0277	4052		std	sinp1	
0300	2226		ldf	nmax	
0301	6104		nzf	aa	
0302	0400	dd	ldn	00	
0303	4052		std	sinp1	
0304	2221		ldf	pmax	
0305	4066	aa	std	prod1	
0306	2010	f	ldd	a2u	
0307	1215		lpf	const	
0310	6036		zjf	j	
0311	2051		ldd	magp2	P_2 overflow test.
0312	3455		sbd	mags2	
0313	6233		pjf	j	
0314	6032		zjf	j	

PROGRAM DIGIT (con't)

Cell	Mach Code	Symbolic			Remarks
0315	2047		ldd	errflg	
0316	0202		lpn	02	
0317	6110		nzf	k	
0320	7500		exf		
0321	2401			2401	Error stop 2 if
0322	0402		ldn	02	P_2 overflows.
0323	0000		err		
0324	7000	const		7000	
0325	3777	pmax		3777	Table of constants.
0326	4000	nmax		4000	
0327	2054	k	ldd	sins2	
0330	6004		zjf	l	
0331	2007		ldd	sina2	
0332	6004		zjf	ff	
0333	6107		nzf	ee	
0334	2007	l	ldd	sina2	P_2 overflow correction.
0335	6005		zjf	ee	
0336	0401	ff	ldn	01	
0337	4050		std	sinp2	
0340	2312		ldb	nmax	
0341	6104		nzf	bb	
0342	0400	ee	ldn	00	
0343	4050		std	sinp2	
0344	2317		ldb	pmax	
0345	4070	bb	std	prod2	
0346	2066	j	ldd	prodl	
0347	3070		add	prod2	Compute
0350	4072		std	sum3	sum3 = $P_1 + P_2$.
0351	2052		ldd	sinpl	
0352	3450		sbd	sinp2	
0353	6125		nzf	m	
0354	2072		ldd	sum3	Determine sign of
0355	6203		pjf	n	sum3 and perform
0356	0401		ldn	01	overflow test.
0357	6102		nzf	o	
0360	0400	n	ldn	00	
0361	3452	o	sbd	sinpl	
0362	6016		zjf	m	
0363	2047		ldd	errflg	
0364	0204		lpn	04	Error stop 3 if
0365	6105		nzf	p	sum3 overflows.
0366	7500		exf		
0367	2401			2401	
0370	0403		ldn	03	
0371	0000		err		

[illegible]

```
Cell   Mach    Symbolic                 Remarks
       Code
0372   2052    p       ldd   sinpl
0373   6003            zjf   q
0374   2346            ldb   nmax       Sum3 overflow correction.
0375   6102            nzf   r
0376   2351    q       ldb   pmax
0377   4072    r       std   sum3
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0400   7500    m       exf
0401   2411                  2411
0402   7325            out   sum        Output sum3.
0403   0073                  73
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0404   2011            ldd   delflg
0405   6013            zjf   zz
0406   2012            ldd   csdel
0407   4046            std   del
0410   2013    ol      ldd   fndel      Delay loop.
0411   0701    il      sbn   01
0412   6501            nzb   il
0413   2046            ldd   del
0414   0701            sbn   01
0415   4046            std   del
0416   6506            nzb   ol
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0417   7075    s       jpi   loop       Get next sample.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0420   7500    zz      exf
0421   1401                  1401
0422   7204            inp   dump       Dump 1st sample if external
0423   0042                  42           timing used.
0424   6505            nzb   s
0425   6406            zjb   s
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0426   0041    dump          41         Input-output table.
0427   0072    sum           72
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       0000    sradd   equ   00
       0005    sinal   equ   05
       0006    alu     equ   06
       0007    sina2   equ   07
       0010    a2u     equ   10
       0011    delflg  equ   11
       0012    csdel   equ   12
       0013    fndel   equ   13
       0040    step    equ   40
       0043    suml    equ   43
       0045    sum2    equ   45         Symbol-cell equivalence
       0046    del     equ   46            table.
       0047    errflg  equ   47
       0050    sinp2   equ   50
       0051    magp2   equ   51
       0052    sinpl   equ   52
       0053    magpl   equ   53
```

Cell	Mach Code	Symbolic			Remarks
	0054	sins2	equ	54	
	0055	mags2	equ	55	
	0056	sins1	equ	56	
	0057	mags1	equ	57	
	0062	onea	equ	62	
	0065	oneb	equ	65	
	0066	prod1	equ	66	
	0070	prod2	equ	70	
	0072	sum3	equ	72	
	0073	rdot	equ	73	
	0074	ramp	equ	74	
	0075	loop	equ	75	
	0076	X1	equ	76	
	0077	X2	equ	77	
	0000		end		